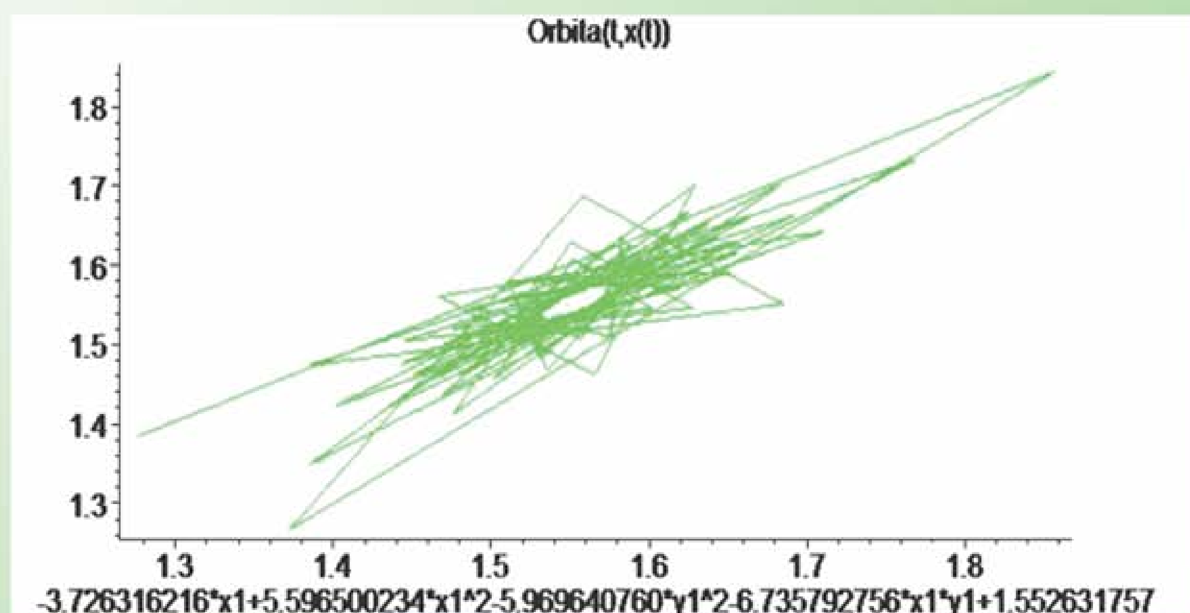
DANIEL N. POP

INIȚIERE ÎN MATLAB

Câteva aplicații practice în diverse domenii de activitate



Presa Universitară Clujeană

DANIEL N. POP

INIȚIERE ÎN MATLAB

CÂTEVA APLICAȚII PRACTICE
ÎN DIVERSE DOMENII DE ACTIVITATE

Referenţi ştiinţifici:

Prof. univ. dr. Eugen Ioan Drăghici

Conf. univ. dr. Radu Trîmbiţaş

ISBN 978-973-595-672-1

Universitatea Babeş-Bolyai
Presa Universitară Clujeană
Director: Codruţa Săcelean
Str. Hasdeu nr. 51
400371 Cluj-Napoca, România
Tel./fax: (+40)-264-597.401
E-mail: editura@editura.ubbcluj.ro
http://www.editura.ubbcluj.ro/

DANIEL N. POP

INIȚIERE ÎN MATLAB

CÂTEVA APLICAȚII PRACTICE ÎN DIVERSE DOMENII DE ACTIVITATE

PRESA UNIVERSITARĂ CLUJEANĂ
2014

Profesorilor mei trecuți în veşnicie.

CUPRINS

PREFAŢĂ

Lucrarea de faţă se adresează, tuturor acelora (elevi, studenţi, ingineri de formaţiuni diferite, medici, economişti etc.) care doresc să înţeleagă rolul softului matematic MATLAB® în activitatea de cercetare şi simulare.
Am utilizat în cuprinsul acestei cărţi ***versiunea*** MATLAB® ***v 7.7 release 2008b.***
Pentru a lărgii în mod natural aria de interes a lucrării, vor fi tratate pe scurt utilizarea acestuia în domeniile:

- Noţiuni introductive în mediul de programare MATLAB®.
- Algebră.
- Analiză matematică.
- Elemente de Teoria Probabilităţilor şi Statistică Matematică.
- Rezolvarea Ecuaţiilor diferenţiale şi cu derivate parţiale.
- Reţele Neuronale.
- Procesarea imaginilor.
- Pachetul de simulare SIMULINK®.

Mare parte din lucrarea de faţă face în mod curent obiectul unor cursuri predate câţiva ani la rând studenţilor Facultăţii de Ştiinţe Economice şi a Facultăţii de Inginerie Economică în domeniile electric, energetic, electronic şi al Facultăţii de Calculatoare din cadrul Universităţii Româno-Germane din Sibiu, precum şi în cadrul Facultăţii de Mecanică secţia S.I.A din cadrul Universităţii Lucian Blaga Sibiu.

Mulţumesc pe această cale domnilor prof. dr. Damian Trif , conf dr. Radu T. Trîmbiţaş de la Universitatea Babeş-Bolyai Cluj Napoca şi prof.dr. Eugen Drăghici de la Universitatea Lucian Blaga din Sibiu pentru sfaturile deosebit de utile, care m-au ajutat enorm în redactarea acestei lucrări.

Orice observaţii sau comentarii, în special critice, sunt bine primite la adresele de e-mail: popdaniel31@yahoo.com, daniel.pop@roger_univ.ro.

CAPITOLUL 1.

PREZENTARE GENERALĂ

Definiţie MATLAB® = Limbaj de înaltă performanţă pentru proiectarea asistată de calculator. El este în acelaşi timp un limbaj de programare şi un sistem de dezvoltare care integrează calculul, vizualizarea şi programarea într-un mediu uşor de utilizat, problemele şi soluţiile acestora fiind exprimate într-un limbaj matematic accesibil.

Domenii de utilizare:

- Matematică şi calcul numeric
- Dezvoltarea algoritmilor
- Modelare, simulare şi testarea prototipurilor
- Analiza şi vizualizarea datelor
- Grafică inginerească şi din ştiinţele aplicate
- Dezvoltarea de aplicaţii, inclusive interfeţe grafice

MATLAB® [7], [15] este un sistem interactiv care are ca element de bază tabloul, matricea, ceea ce permite rezolvarea problemelor de calcul numeric, în special cele care necesită prelucrarea de vectori sau matrici.

Numele MATLAB® ***provine de la Matrix laboratory, firma producătoare este The MathWorks, Inc., SUA.***

MATLAB® -ul a evoluat:

- în mediul universitar unde este pachetul standard pentru cursurile introductive şi avansate de matematică, inginerie, medicină, ştiinţe.
- în industrie, unde este utilizat pentru cercetarea de înalt randament, dezvoltare şi producţie

MATLAB® permite dezvoltarea unei familii de aplicaţii sub forma toolbox-urilor. Aceste toolbox-uri permit învăţarea şi aplicarea tehnologiilor specializate din diverse domenii. Sunt disponibile toolbox-uri pentru domenii cum ar fi: procesarea numerică a semnalelor, sisteme de conducere automată, reţele Neuronale, logică fuzzy, wavelet, simulare (SIMULINK), identificare etc.

Sistemul MATLAB® constă în cinci părţi principale:

- Limbajul MATLAB®.
- Mediul de lucru MATLAB.
- Handle Graphics®.
- Biblioteca de funcţii matematice a MATLAB® -ului.
- Interfaţa de aplicaţii program a MATLAB® -ului (API).

1.1. Limbajul MATLAB

Reprezintă un limbaj de nivel înalt de tip matrice/tablou cu instrucţiuni de control al salturilor, funcţii, structuri de date, intrări/ieşiri şi cu proprietăţi de programare orientată pe obiecte.

Facilităţile de programare sunt organizate pe 6 directoare:

Ops	Operatori şi caractere speciale.
Lang	Comenzi ale limbajului de programare.
Strfun	Şiruri de caractere.
Iofun	Fişiere input/output.
Timefun	Funcţii pentru data calendaristică şi timp.
Datatypes	Tipuri de date şi structuri.

Mediul de lucru MATLAB® reprezintă un set de facilităţi care permit manevrarea variabilelor în spaţiul de lucru, importul şi exportul de date, dezvoltarea, manipularea, editarea şi depanarea fişierelor MATLAB® (.m) şi a aplicaţiilor MATLAB.
Aceste facilităţi sunt organizate în directorul:

general	Comenzi generale.

1.2. Handle Graphics®

Reprezintă sistemul grafic al MATLAB® ului. Cuprinde comenzi de înalt nivel pentru vizualizarea datelor bi şi tri-dimensionale, procesarea imaginilor, animaţie, prezentări de grafice. Permite de asemenea utilizarea unor comenzi de nivel scăzut pentru crearea unor interfeţe grafice GUI. Funcţiile grafice sunt organizate în 5 directoare:

graph2d	Grafice bidimensionale
graph3d	Grafice tridimensionale
specgraph	Grafice ultraspecializate.
Graphics	Unelte grafice.
Uitools	Instrumente de interfaţă grafică pentru utilizator.

Biblioteca de funcţii matematice a MATLAB® -ului: reprezintă o colecţie complexă de algoritmi de calcul pornind de la funcţii elementare (sinus, cosinus etc.) până la funcţii sofisticate (inversarea de matrice, valori proprii, funcţii Bessel, FFT etc.).
Funcţiile matematice sunt organizate în 8 directoare:

Elmat	Operaţii cu matrici.
Elfun	Funcţii matematice elementare.
Specfun	Funcţii matematice speciale.
Matfun	Funcţii de matrici.
Datafun	Transformata Fourier.
Polyfun	Polinoame şi interpolări.
Funfun	Rezolvitori ODE.
Sparfun	Matrici rare.

1.3. Interfaţa de aplicaţii program a MATLAB®-ului (API)

Este o bibliotecă care permite scrierea de programe în **C++** sau **Java** care interacţionează cu MATLAB® -ul. Include facilităţi pentru apelarea rutinelor din MATLAB® apelarea MATLAB® -ului ca maşină de calcul, scrierea şi citirea fişierelor de tip .MAT .

1.4. Toolbox-urile MATLAB®

Toolbox-urile reprezintă o familie de aplicaţii care permit învăţarea şi aplicarea tehnologiilor specializate din diverse domenii. Aceste toolbox-uri sunt colecţii de funcţii MATLAB® (functions) (M-files) care extind mediul MATLAB® pentru rezolvarea unor clase particulare de probleme. Câteva din cele mai utilizate aplicaţii sunt prezentate în figura următoare.

1.5. Pachetul SIMULINK®

SIMULINK® este un pachet software ataşat MATLAB® -ului şi reprezintă un sistem interactiv pentru simularea dinamicii sistemelor neliniare (bineînţeles şi a celor liniare). Este conceput sub forma unei interfeţe grafice care permite crearea unui model prin "trasarea" schemei bloc a sistemului şi apoi simularea dinamicii sistemului.
SIMULINK® poate lucra cu sisteme liniare, neliniare, continue, discrete, multivariabile etc.
SIMULINK® beneficiază de aşa-numitele ***Blockset-uri*** care sunt de fapt biblioteci suplimentare care conţin aplicaţii specializate din domenii cum ar fi: comunicaţii, procesarea semnalelor, bioinformatică, reţele Neuronale etc.
REAL TIME WORKSHOP® este un program foarte important care permite generarea unui ***cod Java*** pentru schemele bloc create în SIMULINK® şi prin urmare permite rularea de aplicaţii în timp real de o mare diversitate.

CAPITOLUL 2.

FUNDAMENTELE PROGRAMĂRII ÎN MATLAB

2.1. Expresii fundamentale

MATLAB® -ul lucrează cu expresii matematice ca şi celelalte limbaje de programare, dar spre deosebire de majoritatea acestor limbaje, aceste expresii implică la scară largă lucrul cu matrici.
Expresiile sunt alcătuite cu ajutorul următoarelor tipuri:

- Variabile
- Numere
- Operatori
- Funcţii

2.1.1. *Variabile*

MATLAB® -ul nu necesită declararea dimensiunii variabilelor, deoarece la întâlnirea unui nou nume de variabilă generează automat variabila respectivă şi alocă spaţiul necesar de memorie.
Numele unei variabile este o literă, urmată de un număr oricât de mare de litere, cifre sau simboluri. Din acest număr "oricât de mare" sunt păstrate primele 31 de caractere.
MATLAB® -ul este **case sensitive** - face distincţie între literele mici şi cele mari.
Exemplu:

```
» a = 85
```

creează o matrice 1 x 1 cu numele a şi stochează valoarea acesteia 85 într-o singură locaţie corespunzătoare singurului element al matricei.

2.1.2. *Numere*

MATLAB® -ul utilizează notaţia zecimală, cu punct zecimal opţional şi cu semn + sau -. Se utilizează şi notaţia ştiinţifică cu litera e pentru a specifica o putere a lui 10. Reprezentarea numerelor imaginare este realizată cu litera i sau j ca sufix.

Exemple:

```
3                  -48              0.00001
5.4873679      1.43267e-15          5.03456e13
3i               -7.348923j              5e3i
```

Toate numerele sunt stocate intern utilizând formatul long specificat de ***standardul IEEE*** în virgulă mobilă (precizie de 16 zecimale semnificative în domeniul 10-308 la 10+308).

Operatori

Expresiile utilizează operatori aritmetici uzuali:

+	Adunare
-	Scădere

*	Multiplicare
/	Împărţire
\	Împărţire la stânga
^	Ridicarea la o putere
'	Transpusa complex conjugată
( )	Operatorul de specificare a ordinii de evaluare

2.1.4 *Funcţii*

MATLAB® -ul furnizează un mare număr de funcţii matematice elementare standard (`abs, sqrt, exp, sin …).`
Există şi funcţii matematice avansate (`funcţii Bessel, Gama etc.`), multe dintre acestea acceptând argumente complexe.
Pentru vizualizarea funcţiilor elementare se poate tasta:

```
» help elfun
```

Pentru a vedea lista funcţiilor avansate se poate tasta:

```
» help specfun
» help elmat
```

O parte din funcţii (cum ar fi sqrt, sin)sunt de tip **built-in**, adică sunt o parte a nucleului MATLAB®, au o mare eficienţă, dar detaliile constructive nu sunt accesibile utilizatorului.
Alte funcţii sunt implementate ca fişiere MATLAB® (M-files) şi pot fi chiar modificate.
Câteva funcţii furnizează valorile unor constante universale:

pi	3.14159265.
I	Sqrt (-1).
J	La fel ca I.
Realmin	Cel mai mic număr în virgulă flotantă, 2-1022
Realmax	Cel mai mare număr în virgulă flotantă, 21023
Inf	Infinit.
NaN	Not-a-number

Numele funcţiilor nu sunt rezervate şi deci este posibilă suprascrierea lor.

Exemplu:

```
eps = 3.e-9
```

Funcţia originală este reconstituită prin comanda:

```
» clear eps
```

2.1.5 *Expresii*

Exemple de expresii şi rezultatele corespunzătoare ale evaluării acestor expresii:

```
» rad=(1+sqrt(12))/2
rad =
     2.2321
» a = abs(12+5i)
a =
     13
» z = sqrt(besselk(4/3,rho-i))
z=
  0.3730+ 0.3214i

» nolin = exp(log(realmax))
nonlin =
  1.7977e+308
» toobig = pi*nonlin
toobig =   Inf
```

2.2. Help on-line, formatul datelor, opţiuni de salvare

2.2.1. Help on-line

Pentru rularea MATLAB® pe un PC trebuie pur şi simplu executat un dublu click cu mouse-ul pe icon-ul MATLAB®. Dacă sistemul de operare nu este de tip Windows (este de tip UNIX) trebuie tastat matlab după prompter-ul sistemului de operare.

Limbajul MATLAB® este mult mai simplu de învăţat, dacă se utilizează comenzile `help`, `helpdesk`, `demo` tastate direct de la prompterul MATLAB®.Pentru aflarea tuturor informaţiilor utile despre o comandă sau o funcţie se tastează help urmat de numele comenzii sau funcţiei respective. Pachetul MATLAB® [15] dispune de asemenea de informaţii complete despre utilizare sub forma unei documentaţii tip .pdf.

În cazuri particulare se poate apela la INTERNET, existând o legătură la pagina Web a firmei producătoare:

`http://www.mathworks.com`

Alte comenzi utile pentru aflarea de informaţii sunt: `helpwin`, `lookfor`, `help help`.

Exemple sugestive de utilizare a comenzii help:

```
» help sin

 SIN    Sine.
    SIN(X) is the sine of the elements of X.
```

```
 Overloaded methods
    help sym/sin.m

» help exp

 EXP    Exponential.
    EXP(X) is the exponential of the elements of X, e
 to the X.
    For complex Z=X+i*Y, EXP(Z) =
 EXP(X)*(COS(Y)+i*SIN(Y)).

    See also LOG, LOG10, EXPM, EXPINT.

 Overloaded methods
    help sym/exp.m
    help demtseries/exp.m

» help plot

 PLOT   Linear plot.

PLOT(X,Y) plots vector Y versus vector X. If X or Y is a matrix, then the
vector is plotted versus the rows or columns of the matrix, whichever line
up. If X is a scalar and Y is a vector, length(Y) disconnected points are
plotted.
PLOT(Y) plots the columns of Y versus their index. If Y is complex,
PLOT(Y) is equivalent to PLOT(real(Y),imag(Y)). In all other uses of PLOT,
the imaginary part is ignored.
Various line types, plot symbols and colors may be obtained with
PLOT(X,Y,S) where S is a character string made from one element from any
or all the following 3 colunms:
           b     blue          .     point              -     solid
           g     green         o     circle             :     dotted
           r     red           x     x-mark             -.    dashdot
           c     cyan          +     plus               --    dashed
           m     magenta       *     star             (none)  no line
           y     yellow        s     square
           k     black         d     diamond
           w     white         v     triangle (down)
                               ^     triangle (up)
                               <     triangle (left)
                               >     triangle (right)
                               p     pentagram
                               h     hexagram

    For example, PLOT(X,Y,'c+:') plots a cyan dotted line with a plus
    at each data point; PLOT(X,Y,'bd') plots blue diamond at each data
    point but does not draw any line.

    PLOT(X1,Y1,S1,X2,Y2,S2,X3,Y3,S3,...) combines the plots defined by
    the (X,Y,S) triples, where the X's and Y's are vectors or matrices
    and the S's are strings.

    For example, PLOT(X,Y,'y-',X,Y,'go') plots the data twice, with a
    solid yellow line interpolating green circles at the data points.

    The PLOT command, if no color is specified, makes automatic use of
    the colors specified by the axes ColorOrder property.  The default
    ColorOrder is listed in the table above for color systems where the
    default is blue for one line, and for multiple lines, to cycle
```

```
    through the first six colors in the table.  For monochrome systems,
    PLOT cycles over the axes LineStyleOrder property.

    If you do not specify a marker type, PLOT uses no marker.
    If you do not specify a line style, PLOT uses a solid line.

    PLOT(AX,...) plots into the axes with handle AX.

    PLOT returns a column vector of handles to lineseries objects, one
    handle per plotted line.

    The X,Y pairs, or X,Y,S triples, can be followed by
    parameter/value pairs to specify additional properties
    of the lines. For example, PLOT(X,Y,'LineWidth',2,'Color',
[.6 0 0])
    will create a plot with a dark red line width of 2 points.

    Example
       x = -pi:pi/10:pi;
       y = tan(sin(x)) - sin(tan(x));
       plot(x,y,'--rs','LineWidth',2,...
                       'MarkerEdgeColor','k',...
                       'MarkerFaceColor','g',...
                       'MarkerSize',10)

    See also plottools, semilogx, semilogy, loglog, plotyy, plot3, grid, ti-
tle, xlabel, ylabel, axis, axes, hold, legend, subplot, scatter.
    Overloaded methods:
        timeseries/plot
        phytree/plot
        commscope.plot
        channel.plot
        cfit/plot
        darray/plot
        fints/plot
        idmodel/plot
        idfrd/plot
        iddata/plot
        idnlhw/plot
        idnlarx/plot
        cgrules/plot
        xregtwostage/plot
        xregtransient/plot
        xregmodel/plot
        xregarx/plot
        localmulti/plot
        localmod/plot
        localavfit/plot
        sweepset/plot
        mdevtestplan/plot
        cgdatasetnode/plot
        mpc/plot
        rfckt.plot
        frd/plot
        dspdata.plot
        wdectree/plot
        ntree/plot
        dtree/plot
        wvtree/plot
```

```
        rwvtree/plot
        edwttree/plot

    Reference page in Help browser
        doc plot
```

De exemplu, `Plot (X,Y,` 'c+:') desenează o linie punctată de culoare cyan cu un marker + în fiecare punct al graficului. Dacă comanda `Plot` nu are culori specificate, atunci preia implicit culorile axelor.

2.2.2. Formatul datelor

MATLAB® -ul afişează numerele cu 5 zecimale (setare implicită). Această setare se poate modifica cu ajutorul comenzii format:

```
>>FORMAT Set output format.
All computations in MATLAB are done in double precision.
    FORMAT may be used to switch between different output
    display formats as follows:
    FORMAT          Default. Same as SHORT.
    FORMAT SHORT    Scaled fixed point format with 5 digits.
    FORMAT LONG     Scaled fixed point format with 15 digits.
    FORMAT SHORT E  Floating point format with 5 digits.
    FORMAT LONG E   Floating point format with 15 digits.
    FORMAT SHORT G  Best of fixed or floating point format with
       5 digits.
    FORMAT LONG G   Best of fixed or floating point format with
       15 digits.
     FORMAT HEX      Hexadecimal format.
    FORMAT +        The symbols +, - and blank are printed
                      for positive, negative and zero elements.
                      Imaginary parts are ignored.
    FORMAT BANK     Fixed format for dollars and cents.
    FORMAT RAT      Approximation by ratio of small integers.
    Spacing:
    FORMAT COMPACT Suppress extra line-feeds.
    FORMAT LOOSE    Puts the extra line-feeds back in.
```

Exemple:

```
» c=1.333456789233
c =
    1.3335

» format long
» c
c =
   1.33345678923300

» format short e
» c
c =
  1.3335e+000
```

```
» format long e
» c
c =
    1.333456789233000e+000

» format
» c
c =
    1.3335
```

2.2.3. Opţiuni de salvare

Pentru salvarea variabilelor curente cu care se lucrează în MATLAB la încheierea unei sesiuni de lucru se poate utiliza comanda **save**, această comandă va salva toate variabilele curente generate de către utilizator într-un fişier numit **matlab.mat** . Dacă se doreşte se poate da un nume fişierului de date în care se salvează variabilele.

Exemplu:

```
» save date c determ A%
```

realizează salvarea datelor c, determ şi A într-un fişier date.mat .
Pentru restituirea variabilelor într-o sesiune de lucru ulterioară se foloseşte comanda **load.**
Exemplu:

```
» load date
```

Dacă se doreşte aflarea variabilelor curente se pot utiliza comenzile who,whos:

```
» who
Your variables are:

A         c         determ

» whos
  Name              Size                Bytes  Class

  A                 2x2                32  double array
  c                 1x1                 8  double array
  determ            1x1                 8  double array
is 6 Grand total elements using 48 bytes.
```

Pentru ştergerea tuturor variabilelor curente din memoria de lucru se poate utiliza comanda **clear.**

2.3. Crearea fişierelor MATLAB (.m files)

Deoarece este mult mai comod şi util decât introducerea comenzilor linie după linie la prompterul MATLAB®, se lucrează cu fişiere text care conţin aceste linii program cu comenzile necesare.
Aceste fişiere conţin cod în limbajul MATLAB® şi sunt denumite .m files (sau M-files). Fişierele se creează utilizând un editor de text şi apoi se utilizează ca o comandă MATLAB® obişnuită (Placeholder1) [16].
Sunt două tipuri de fişiere .m:

- **Fişiere Script**, care nu acceptă argumente de intrare şi nu returnează argumente de ieşire. Aceste fişiere operează cu datele din spaţiul de lucru.
- **Rutine (funcţii),** care acceptă argumente de intrare şi returnează argumente de ieşire. Variabilele utilizate sunt variabile locale (interne) ale funcţiei.

Pentru a vedea conţinutul unui fişier MATLAB®, ***de exemplu*** `balanta.m`, se foloseşte comanda:

```
» type balanta
```

2.3.1. Fişiere Script

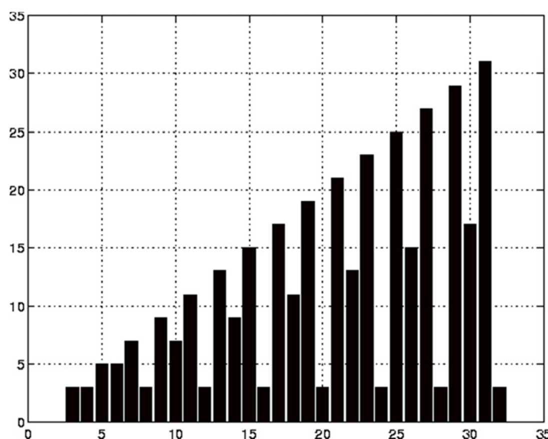
Atunci când se apelează la un fişier script, MATLAB® -ul execută comenzile găsite în fişierul respectiv. Fişierele script pot lucra cu date din spaţiul de lucru (workspace) sau pot crea date noi cu care operează. Script-urile nu furnizează argumente de ieşire, iar variabilele create rămân în workspace, pentru a fi eventual folosite în calculele ulterioare.
Fişierele script pot furniza ieşiri grafice folosind funcţii cum ar fi plot, bar.
Exemplu de fişier script [7] : `magicrank.m`, cu următoarele comenzi MATLAB®:

```
% Investigate the rank of magic squares
r = zeros (1,32);
for n = 3:32
   r(n) = rank(magic(n));
end
r
bar(r)
```

La tastarea numelui fişierului script (fără extensia .m):

```
» magicrank
```

MATLAB® -ul execută comenzile, calculează rangul unor matrici (matricile magice), şi trasează graficul cu rezultatele calculului. După ce se termină execuţia fişierului, variabilele n şi r rămân în spaţiul de lucru.
Graficul rezultat este prezentat în continuare:



2.3.2. Funcţii (rutine)

Aceste fişiere acceptă argumente de intrare şi furnizează argumente de ieşire. Numele fişierului MATLAB® (M-file) şi cel al funcţiei (subrutinei) respective trebuie să fie identice. Funcţiile (subrutinele) lucrează cu variabile proprii separate de spaţiul de lucru uzual al MATLAB® -ului.

Exemplu: funcţia `rank`.
Fişierul M-file rank.m este disponibil în directorul
`toolbox/matlab/matfun`
Se poate vizualiza fişierul cu comanda:

```
» type rank
function r = rank(A,tol)
% RANK Matrix rank.
% RANK(A) provides an estimate of the number of
% linearly independent rows or columns of a matrix A
% RANK(A,tol) is the number of singular values of A
% that are larger than tol.
% RANK(A) uses the default
% tol = max(size(A)) * norm(A) * eps.
s = svd(A);
if nargin==1
   tol = max(size(A)) * max(s) * eps;
end
r = sum(s > tol);
```

Prima linie a unei funcţii M-file începe cu cuvântul cheie ***`function`***. Această linie dă numele funcţiei, ordinea şi numărul argumentelor.
Liniile următoare (care încep cu `caracterul %`) sunt linii de comentariu, care de fapt sunt şi liniile afişate atunci când se apelează la comanda
`» help rank`
Restul liniilor sunt executabile. Variabila `s`, ca şi `r`, `A`, `tol` sunt variabile locale ale funcţiei şi sunt separate de variabilele din workspace.
Funcţia `rank` poate fi utilizată în diferite moduri:

```
» rank(A)
» r = rank(A)
» r = rank(A,1.e-6)
```

2.3.3. Variabile globale

Dacă se doreşte ca mai multe astfel de subrutine să utilizeze o anume variabilă comună, se declară variabila respectivă ca globală utilizând comanda global în toate funcţiile respective.

Exemplu fişierul exg.m:

```
function h = exg(t)
global GB
h = 1/2*GB*t.^2;
Se introduc apoi în mod interactiv liniile:
» global GB
» GB = 32;
» y = exg((0:.1:5)');
```

2.3.4. Funcţia eval

Funcţia ***`eval`*** lucrează cu variabilă text pentru implementarea unei facilităţi puternice de tip macro text.
Expresia `eval(s)` foloseşte interpretor-ul MATLAB® pentru evaluarea expresiei sau execuţia declaraţiei din şirul de caractere.

2.3.5. Vectorizarea

Pentru a obţine o viteză de calcul mare, este foarte importantă aşa-numita vectorizare a algoritmilor în fişierele MATLAB®. Acolo unde alte limbaje folosesc bucle de tip for sau `DO`, `MATLAB`-ul poate utiliza operaţii matriceale sau vectoriale.

Un exemplu simplu este următorul:

```
x = 0;
for k = 1:1001
   y(k) = log10(x);
   x = x + .01;
end
```

Versiunea vectorizată a aceluiaşi program este :

```
x = 0:.01:10;
y = log10(x);
```

Atunci când nu se poate elimina complet folosirea unei bucle se utilizează procedura de prealocare.

2.3.6. Funcţii de funcţii

În MATLAB® există o clasă de funcţii care lucrează cu funcţii neliniare ca argument.
Funcţiile de funcţii includ:

- Găsirea zerourilor
- Optimizare
- Integrare numerică
- Ecuaţii diferenţiale ordinare

MATLAB® -ul reprezintă funcţia neliniară ca o funcţie M-file care poate fi ulterior utilizată ca argument de alte funcţii MATLAB®.

Exemplu:

Următorul fişier creează o funcţie neliniară:

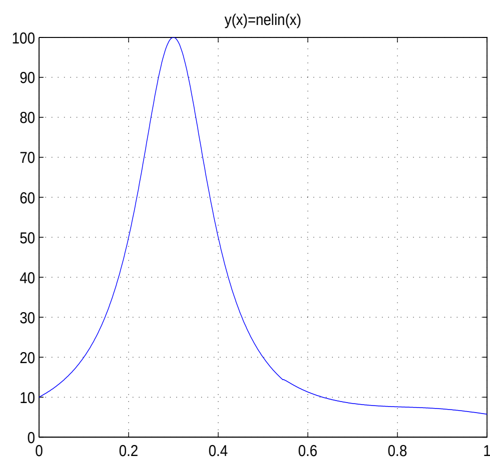
```
function y = nelin(x)
y = 1./((x-.3).^2 + .01) + sqrt(1./((x-.9).^2 + .04) - 6);
```

Această funcţie poate fi evaluată pentru un set de puncte în intervalul 0 <=x <=1 cu programul:

```
x = 0:.002:1;
y = nelin(x);
```

şi apoi se poate reprezenta grafic funcţia cu comanda:

```
plot(x,y); grid on; title('y=nelin(x)')
```



Dacă dorim să evaluăm această funcţie în

```
>>p =0.637;
    0.637
>> nelin(p);
ans =
   5.1192
```

CAPITOLUL 3.

MEDIUL DE LUCRU MATLAB

MATLAB® [16] este, după cum s-a afirmat deja, atât un limbaj cât şi un mediu de programare. Ca mediu de lucru, MATLAB® include facilităţi pentru manipularea variabilelor în spaţiul de lucru, pentru importul şi exportul datelor, precum şi instrumente pentru dezvoltarea şi manipularea fişierelor (M-files) şi a aplicaţiilor MATLAB®.
Mediul de programare este utilizat în mod diferit în funcţie de platforma pe care rulează MATLAB® -ul (este vorba de sistemul de operare care poate fi de tip Windows , Linux, Unix).

3.1. Lansarea sesiunilor de lucru

Pe platformele de tip Windows, programul de instalare creează un "shortcut" la programul executabil, shortcut care poate fi plasat pe desktop (pe display-ul de lucru al calculatorului). Prin efectuarea unui dublu click pe icon-ul care reprezintă acest shortcut se startează MATLAB® -ul.
Pentru startarea MATLAB® -ului pe un sistem UNIX trebuie tastat matlab la prompterul sistemului de operare.

3.1.1. Fişiere de pornire (Startup Files)

La pornireMATLAB® -ul execută automat **`fişierul master matlabrc.m`** şi, dacă există, **`fişierul startup.m.`**
Fişierul matlabrc.m este rezervat pentru administratorul de sistem (reţea), în timp ce fişierul startup.m este destinat utilizatorilor. De aici se pot seta căile de acces, se pot defini setările implicite pentru instrumentele Handle Graphics şi se pot predefini variabile în spaţiul de lucru.
De exemplu, dacă în startup.m se introduce linia:

```
addpath /home/me/mytools
```

se adaugă un director de instrumente proprii pentru calea implicită de căutare.
Pe platformele Windows, fişierul **startup.m** se plasează în directorul local din directorul **toolbox.**

3.1.2. Opţiuni de pornire

Se pot specifica opţiuni de pornire, aceste opţiuni fiind adăugate pe calea shortcut-ului MATLAB.
În continuare sunt prezentate câteva posibile opţiuni.

Opţiunea de Startup	Descriere
Automation	Lansează MATLAB-ul ca un server automat, minimizat, fără "splash screen".
Logfile log-filename	Scrie în mod automat ieşirile din MATLAB într-un "log file" specificat.
Minimize	Lansează MATLAB-ul minimizat şi fără "splash screen"-ul MATLAB.
Nosplash	Startează MATLAB-ul fără afişarea "splash screen"-ului MATLAB.
r M_file	Rulează automat fişierul .m specificat imediat după lansarea MATLAB.ului.
Regserver	Modifică regiştrii Windows cu intrări adecvate tip ActiveX pentru MATLAB.
Unregserver	Modifică regiştrii Windows pentru ştergerea intrărilor ActiveX. Se utilizează pentru resetarea regiştrilor.

De exemplu, pentru a porni MATLAB® -ul şi a rula imediat în mod automat un fişier cum ar fi de exemplu rezultate.m, se poate utiliza următoarea cale pentru shortcut:
`C:\bin\nt\matlab.exe /r rezultate`

3.1.3. Terminarea unei sesiuni de lucru

Pentru a părăsi mediul MATLAB, se tastează ***`quit`*** `la` prompterul MATLAB®.
În cazul platformelor Windows, se poate termina sesiunea prin selectarea opţiunii exit din meniul File sau se poate utiliza butonul close vizibil în colţul din dreapta sus al ferestrei de comandă MATLAB®.
Se pot include comenzi de tipul save în acest fişier, pentru a salva automat variabilele din spaţiul de lucru la încheierea sesiunii. O altă opţiune este de a cere afişarea unei casete de dialog pentru confirmarea terminării sesiunii.
Aceste două tipuri de opţiuni se pot găsi în **directorul /toolbox/local:**
`finishsav.m`: permite salvarea variabilelor curente
`finishdlg.m`: afişează o casetă de dialog pentru confirmarea opririi

3.2. Fereastra de comandă (fereastra principală)

Fereastra de comandă este fereastra principală prin intermediul căreia se poate comunica cu MATLAB-ul.

 Pe platformele Windows, MATLAB-ul furnizează o fereastră specială, cu facilităţi de tip Windows.

 Pe sistemele UNIX, fereastra de comandă este fereastra terminal din care este lansat MATLAB-ul.

Interpreterul MATLAB afişează un prompter (>>) indicând faptul că este gata să accepte comenzile utilizatorului. De exemplu, pentru introducerea unei matrici 3 x 3 se poate tasta:
» A = [1 2 3; 4 5 6; 7 8 10]

şi la apăsarea tastelor Enter sau Return, MATLAB-ul răspunde cu:

```
A =
      1       2       3
      4       5       6
      7       8      10
```

Editarea liniilor de comandă în fereastra principală
Tastele de tip săgeată şi tasta ***Ctrl*** permit apelarea, editarea şi eventual reutilizarea comenzilor editate anterior. De exemplu:

```
» rad = (1+ sqt(5))/2
```

Undefined function or variable 'sqt'.
Pentru eliminarea greşelii de editare a numelui funcţiei radical (sqrt) nu se mai editează din nou toată linia, ci se foloseşte tasta ↑, apare din nou linia de comandă greşită şi apoi cu tasta ← se deplasează cursorul pe linie şi se repară greşeala.
Lista completă a săgeţilor şi tastelor care permit controlul asupra operaţiunilor cu linia de comandă este prezentată în tabelul următor:

Arrow Key	Control Key	Operation
↑	Ctrl-p	Recall previous line.
↓	Ctrl-n	Recall next line.
←	Ctrl-b	Move back one character.
→	Ctrl-f	Move forward one character.
ctrl- →	Ctrl-r	Move right one word.
ctrl- ←	Ctrl-l	Move left one word.
home	Ctrl-a	Move to beginning of line.
end	Ctrl-e	Move to end of line.
esc	Ctrl-u	Clear line.
del	Ctrl-d	Delete character at cursor.
backspace	Ctrl-h	Delete character before cursor.
	Ctrl-k	Delete (kill) to end of line.

3.2.1. Ştergerea ferestrei de comandă

Pentru ştergerea conţinutului (afişajul) ferestrei principale se poate folosi comanda `clc`, **care însă nu are ca efect ştergerea variabilelor curente din spaţiul de lucru.**

3.2.2. Controlul afişării paginilor ecran în fereastra de comandă

Pentru a controla afişarea paginilor în fereastra de comandă se utilizează comanda more. Setarea implicită este more off. Atunci când setăm `more on`, o pagină ecran este afişată. Apoi se poate utiliza:

Return	To advance to the next line
Space Bar	To advance to the next page
q	To stop displaying the output

Întreruperea unui program care rulează
Se poate întrerupe un program care rulează prin apăsarea pe tastele **`Ctrl+c.`**

 Pe sistemele Windows se va aştepta terminarea execuţiei funcţiilor de tip built-in sau a fişierelor de tip MEX-file.

 Pe sistemele UNIX, execuţia programului se va încheia imediat.

Comanda format
Comanda format controlează formatul numeric al valorilor afişate pe ecran şi a fost deja discutată într-un paragraf anterior. Comanda are efect asupra afişării numerelor, şi nu asupra modului intern de reprezentare a acestora.
Pe sistemele Windows, se poate schimba setarea implicită a formatului prin selectarea meniului Preferences din meniul File şi selectarea formatului dorit din General.

3.2.3. Suprimarea afişării rezultatelor unei linii comandă

Deoarece la apăsarea tastelor **Return** sau **Enter** **MATLAB**-ul afişează automat rezultatele pe ecran, dacă încheiem linia de comandă cu punct şi virgulă, va fi realizat calculul, dar nu va mai fi afişat. Această facilitate este importantă atunci când avem de lucrat cu matrici mari sau cu multe date.

Exemplu:
```
A = magic(100);
```

3.2.4. Linii de comandă lungi

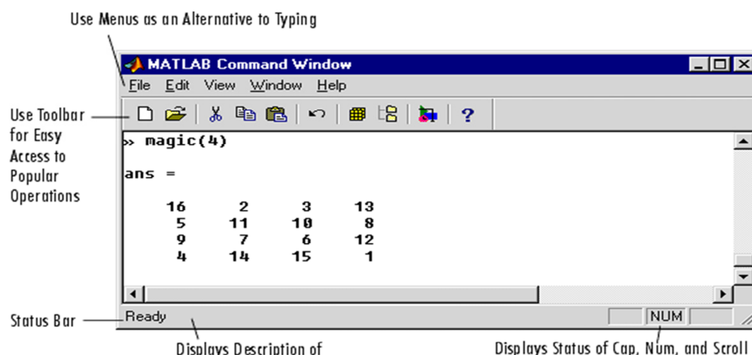
Dacă o declaraţie nu încape într-o linie de comandă, se pot utiliza trei puncte urmate de Return sau Enter pentru a indica faptul că expresia continuă pe linia următoare.
Exemplu:

```
s = 1 - 1/2 + 1/3 - 1/4 + 1/5 - 1/6 + 1/7 ...
      - 1/8 + 1/9 - 1/10 + 1/11 - 1/12;
```

Spaţiile albe din jurul semnelor =, +, - sunt opţionale şi pot îmbunătăţi citirea liniilor. Maximul numărului de caractere pe o singură linie este de 4096.

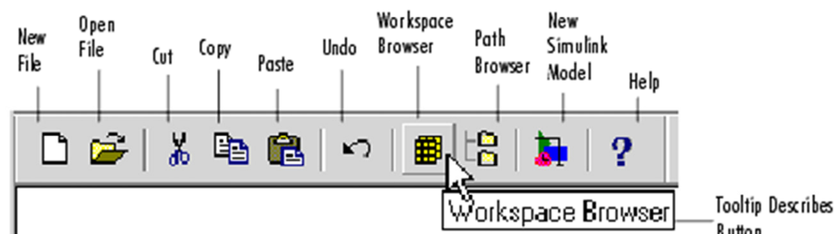
3.2.5. Descrierea ferestrei de comandă

Fereastra de comandă permite rularea comenzilor MATLAB®, lansarea unor instrumente cum ar fi ***Editor/Debugger*** şi permite startarea toolbox-urilor.



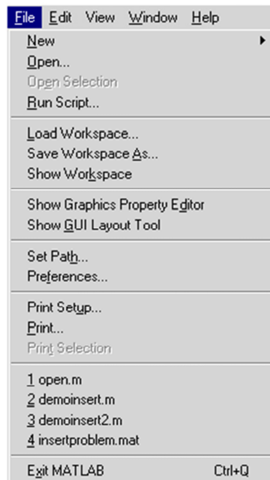
3.2.6. Toolbar (bara de instrumente)

Toolbar-ul din fereastra de comandă permite accesul la operaţii simple:



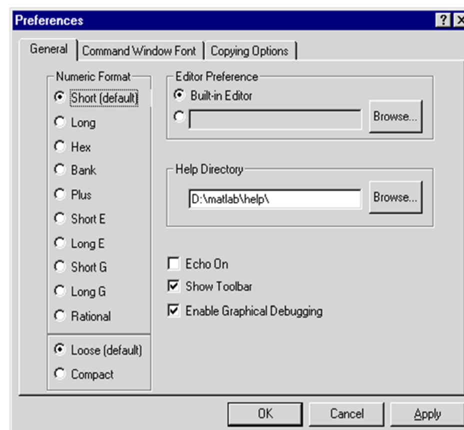
3.2.7. Meniuri

Meniurile ferestrei de comandă furnizează accesul la operaţii care nu sunt accesibile din ***toolbar***.



3.2.8. Preferinţe

Sunt utilizate pentru controlarea operaţiilor şi a modului de prezentare din fereastra de comandă. Pentru aceasta se selectează **Preferences** din meniul **File**, iar după apariţia casetei de dialog **Preferences** se pot seta preferinţele pentru **General, Command Window Font şi Copying Options.**



General Preferences:

- `Numeric Format` – Specifică formatul numeric implicit.
- `Editor Preference` – Permite folosirea editorului MATLAB® sau specificarea altui editor.
- `Help Directory` – Specifică directorul în care se află fişierele de tip help.
- `Echo On` – Setează facilitatea de afişare a liniilor program în timp ce un program MATLAB® este rulat.

- `Show Toolbar` – Arată sau ascunde toolbar-ul.
- `Enable Graphical Debugging` – Permite depanarea (Debugger) în mod automat la fiecare breakpoint.
- `Command Window Font`-Specifică caracteristicile fontului pentru textul afişat în fereastra de comandă.
- `Copying Options`: -Specifică opţiunile utilizate la copierea unor obiecte din MATLAB® în clipboard pentru rescrierea în alte aplicaţii.

3.2.9. Spaţiul de lucru al MATLAB-ului (workspace)

Spaţiul de lucru conţine un set de variabie (numite tablouri sau matrice) care pot fi manevrate din linia de comandă. Se pot folosi comenzile **`who`** şi **`whos`** (deja prezentate) pentru a vedea care sunt variabilele curente din workspace.
Pentru ştergerea variabilelor din workspace se utilizează comanda **`clear.`**

Încărcarea şi salvarea din workspace

Comenzile save şi load descrise într-un subcapitol anterior au rolul de a salva variabilele din spaţiul de lucru şi respectiv de a le reîncărca într-o sesiune ulterioară. Aceste comenzi se pot folosi şi pentru a importa şi exporta date.

Pe platformele Windows, operaţiile save,load sunt disponibile şi prin selectarea opţiunilor Save Workspace As, respectiv Load Workspace din meniul File.

Citirea sau scrierea unor fişiere .mat din programe externe în C++ sau Java se poate face cu Interfaţa de Aplicaţii Program (API).
Formatul în care comanda save stochează datele poate fi controlat prin adăugarea unor flaguri în lista de nume de fişiere sau variabile:

-mat	Utilizează formatul binar tip .MAT (implicit).
-ascii	Utilizează formatul ASCII pe 8 digiţi.
-ascii-double	Utilizează formatul ASCII pe 16 digiţi.
-ascii -double –tabs	Delimitează elementele tablourilor cu tab-uri.
-v11	Salvează într-un format pe care versiunea 11 MATLAB îl poate deschide.
-append	Adaugă datele într-un fişier .MAT existent.

Observaţie: Atunci când se salvează conţinutul spaţiului de lucru în format ASCII trebuie salvată câte o variabilă pentru a permite reconstituirea ulterioară a acesteia.

3.2.10. Încărcarea unor fişiere cu date ASCII

Comanda `load` permite importul de fişiere de date ASCII. Exemplu:

```
» load informatii.dat
```

creează o variabilă cu numele diverse în workspace. Dacă datele ASCII au m linii cu n valori pe fiecare linie, rezultatul va fi o matrice numerică m x n.

Nume de fişiere ca şiruri de caractere

Dacă numele fişierelor sau variabilelor cu care se lucrează sunt stocate în variabile de tip **şir de caractere**, se poate folosi dualitatea comandă/funcţie pentru a apela load şi save ca funcţii.

De exemplu:

```
» save('fis1','VAR1','VAR2')
» A = 'fis1';
» load(A)
```

au acelaşi efect ca

```
» save fis1 VAR1 VAR2
» load A
```

Pentru încărcarea sau salvarea mai multor fişiere cu acelaşi prefix şi cu sufixe numere întregi succesive se poate utiliza o buclă.

Exemplu:

```
file = 'prog';
for i = 1:10
    j = i.^2;
    save([file int2str(i)],'j');
end
```

Wildcards

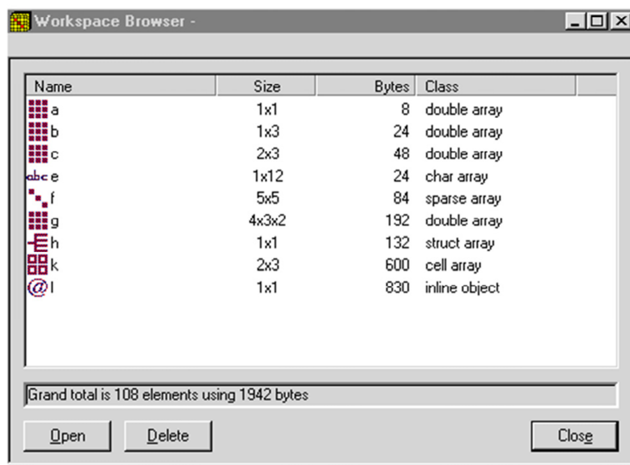
Comenzile `load` şi `save` permit specificarea unui caracter special de tip wildcard (*).

Exemplu:

`» save data x*%`salvează toate variabilele din workspace care încep cu x în fişierul **`data.mat`**.

3.2.11. Browser-ul Workspace

Browser-ul Workspace permite vizualizarea conţinutului spaţiului de lucru curent (este de fapt varianta grafică a comenzii `whos`). Pentru a deschide acest instrument, se selectează Show Workspace din meniul **File** şi apoi se face click din mouse pe tasta **Workspace Browser din toolbar.**



Căi de căutare
MATLAB® -ul utilizează o cale de căutare (search path) pentru a găsi fişierele .m . Aceste fişiere sunt organizate în directoare, din care unele sunt furnizate de MATLAB® şi altele sunt disponibile separat ca toolbox-uri.
Dacă de exemplu tastăm numele `fis1` la prompterul >>, interpretorul MATLAB va face următoarea căutare:

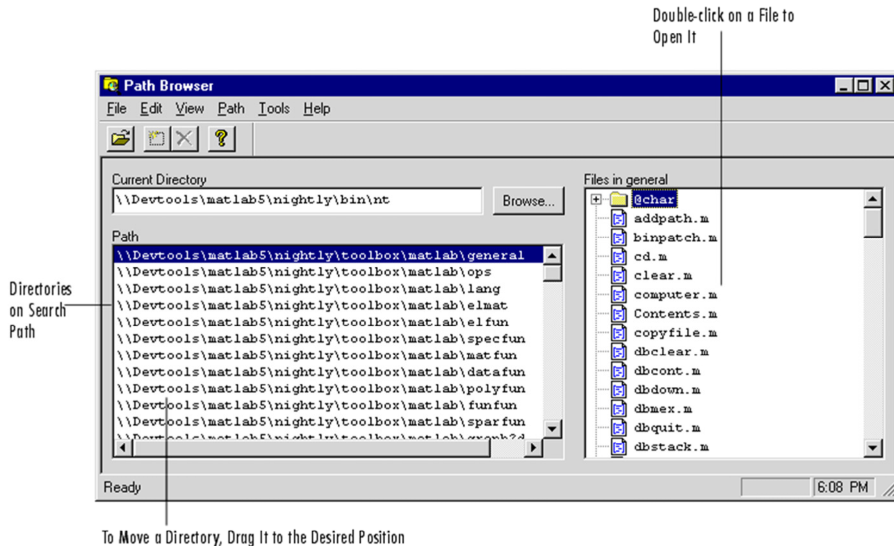
- Caută pe fis1 ca pe o variabilă.
- Verifică dacă fis1 este o funcţie tip `buit-in`.
- Caută în directorul curent fişierul numit `fis1.m`.
- Caută în directoarele aflate în calea de căutare fişierul **fis1.m**.

Schimbarea căii de căutare
Calea de căutare poate fi afişată sau schimbată folosind funcţiile `path, addpath, rmpath`:

- `Path` determină reîntoarcerea la calea curentă.
- `path(s)`, unde s este un şir de caractere, setează calea la s.
- `addpath /home/lib` şi path (`path,'/home/lib'`) adaugă noi directoare la calea de căutare.
- `rmpath /home/lib` şterge calea `/home/lib`.

Calea de căutare implicită este definită în fişierul pathdef.m în directorul local.
Fişiere pe calea de căutare
Pentru a afişa calea de căutare se poate folosi path. Dacă dorim să vedem ce fişiere MATLAB® sunt într-un director se utilizează comanda what. Exemplu:
```
» what matlab/elfun
```
Pentru a vedea conţinutul unui fişier se foloseşte comanda type (deja descrisă într-un paragraf anterior). Exemplu:
```
» type fis2
```
Pentru editarea unui fişier M-file se poate utiliza edit. Exemplu:
```
» edit fis2
```

Path Browser
MATLAB-ul furnizează şi un browser al căii de căutare (Path Browser) cu o interfaţă grafică pentru vizualizarea şi schimbarea căii de căutare [15]. Pentru startarea acestui browser se utilizează `pathtool`, sau se selectează `Set Path` din meniul `File`, sau se face click pe butonul `Path Browser` din toolbar.



Meniurile din **Path Browser** pot fi folosite pentru:

- Adăugarea unui director pe calea de căutare.
- Ştergerea (îndepărtarea) unui director din cale.
- Salvarea setărilor în fişierul pathdef.m.
- Restabilirea setărilor implicite.

3.2.12. Directorul curent

MATLAB® -ul menţine un director curent pentru lucrul cu fişiere de tip *.m şi *.mat.

Pe platformele Windows, directorul curent iniţial este specificat în shortcut-ul utilizat pentru pornirea MATLAB® -ului. Pentru schimbarea setărilor implicite se poate face click cu butonul din dreapta al mouse-ului şi se selectează ***meniul Properties.***

Deschiderea fişierelor în MATLAB
Se pot deschide fişiere în funcţie de extensiile lor prin folosirea funcţiei ***open,*** care este o funcţie extensibilă de către utilizator.
Se pot include şi alte tipuri de fişiere în afara fişierelor MATLAB® standard:

Nume	Acţiune
Figure file (*.fig)	Deschide o figură într-o fereastră tip figură.
M-file (daniel.m)	Deschide fişierul daniel de tip .m în Editor.
Model (daniel.mdl)	Deschide modelul daniel.m în Simulink.
P-file (daniel.p)	Deschide fişierul corespunzător daniel.m, dacă există, în Editor.
Variable	Deschide tabloul name în Array Editor (tabloul trebuie să fie numeric); open apelează openvar.
Alte extensii (daniel.custom)	Deschide daniel.custom prin apelarea funcţiei helper opencustom, unde opencustom este o funcţie definită de utilizator.

3.3. Fereastra grafică (figure)

MATLAB® -ul direcţionează ieşirile grafice spre o fereastră distinctă de fereastra de comandă. Această fereastră grafică este denumită figură (**figure**).
Funcţiile grafice creează în mod automat o nouă fereastră grafică dacă nu există una curentă. Dacă există o astfel de fereastră MATLAB® -ul o va utiliza.
Dacă există mai multe ferestre de tip figură, atunci una dintre ele este asignată ca fiind fereastra curentă (în general este ultima fereastră figură utilizată).
Funcţia figure generează ferestre grafice. De exemplu,

```
>> figure
```

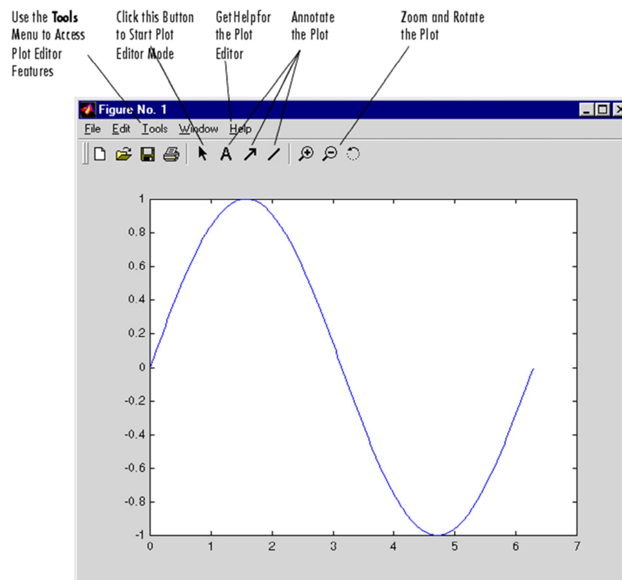
generează o nouă fereastră şi o face fereastra curentă.
O funcţie grafică, cum ar fi funcţia plot, generează un grafic în fereastra de tip figură. De exemplu,

```
>>figure (1)
>>t = 0:pi/100:2*pi;
y = cos(t);
plot(t,y)
```

trasează graficul funcţiei cosinus de la 0 la 2π în fereastra curentă de tip figură, dacă aceasta există, iar dacă nu într-una nou creată.

3.3.1. Prelucrarea graficelor cu Plot Editor

După generarea unui grafic (plot), se pot face schimbări şi prelucrări ale graficului cu interfaţa grafică Plot Editor. Figura următoare ilustrează principalele facilităţi ale ferestrei grafice şi ale interfeţei Plot Editor.
Pentru salvarea unei figuri se selectează ***Save*** din meniul ***Fil***e. Pentru salvarea într-un format diferit, cum ar fi ***EPS,*** necesar utilizării în alte aplicaţii se selectează ***Expor***t din meniul ***File.***



3.4. Importul şi exportul de date

Sunt multe posibilităţi de a realiza importul şi exportul de date între MATLAB şi alte aplicaţii.
În majoritatea cazurilor se pot utiliza facilităţile MATLAB de a citi sau scrie fişiere (pentru aplicaţii complicate trebuie scrise programe în Java).

3.4.1. Importul de date

În tabelul următor sunt prezentate câteva metode de import date:

Metoda	Când trebuie utilizată metoda.
Introducerea unei liste explicite de elemente	Atunci când cantitatea de date este mică. Se tastează pur şi simplu datele utilizând parantezele drepte ([ ]).
Crearea de date într-un fişier .m	Se utilizează un editor de text pentru generarea unui fişier .m. Metoda este utilă atunci când datele nu sunt deja în formă digitală. Este într-un fel similară cu prima metodă.
Încărcarea datelor dintr-un fişier ASCII	Fişierele ASCII stochează datele pe linii cu un număr egal de elemente spaţiate prin blanc-uri, linii încheiate cu Enter. Aceste fişiere se pot edita cu un editor de texte obişnuit. Datele sunt introduse în MATLAB cu funcţia load. Se poate utiliza dlmread dacă este necesară specificarea altor delimitatori.

Citirea datelor cu fopen, fread şi cu funcţiile de intra-re/ieşire	Metoda este folosită când se încarcă date de la alte aplicaţii, date care au propriul lor format.
Funcţii spe-cializate de citire a fişierelor	Dlmread - Citeşte fişiere de date ASCII.
	Wk1read - Citeşte fişiere de tip (WK1) (tip foaie de lucru)
	Imread - Citeşte din fişiere grafice.
	Auread - Citeşte fişiere de sunet tip (.au).
	Waveread - Citeşte fişiere de sunet Microsoft WAVE (.wav).
Crearea de fişiere tip MEX pentru citirea datelor	Este metoda potrivită dacă sunt deja disponibile rutine C++ sau Java pentru citirea datelor din alte aplicaţii.
Dezvoltarea unor pro-grame în Java sau C++	Se utilizează în cazuri complexe pentru translatarea unor date în format .mat şi apoi încărcarea cu comanda load.

3.4.2. Exportul datelor

În tabelul următor sunt prezentate câteva metode de export date:

Metoda	Mod de utilizare	
Folosirea comenzii diary	Pentru tablouri de date de mică dimensiune se foloseşte comanda diary pentru crearea unui fişier de tip jurnal şi afişarea variabilelor. Ieşirea diary include comenzile MATLAB folosite într-o sesiune de lucru.	
Salvarea datelor în for-mat ASCII	Se utilizează comanda save cu opţiunea -ascii. Se poate folosi dlmwrite dacă este necesară specificarea altor delimitatori.	
Scrierea datelor în for-mate speciale	Se folosesc fwrite şi alte funcţii I/O de nivel scăzut. Este utilă la scrierea datelor în formate cerute de alte aplicaţii.	
Funcţii special-izate de scriere a fişierelor	Dlmwrite	Scrie fişiere în format ASCII.
	wk1write	Scrie fişiere tip (WK1).
	Imwrite	Scrie imagini pentru fişiere grafice.
	Auwrite	Scrie fişiere de sunet tip (.au).
	Wavwrite	Scrie fişiere de sunet tip Microsoft WAVE (.wav).

Crearea unor fişiere tip MEX pentru scrierea datelor	Este metoda potrivită dacă sunt deja disponibile rutine C++ sau Java pentru scrierea datelor în formate cerute de alte aplicaţii.
Scrierea datelor în fişiere tip .MAT	Se foloseşte comanda save şi apoi se scrie un program în Java sau C++ pentru translatarea fişierului .mat în formatul dorit.

3.4.3. Fişiere de tip text cu delimitatori

Funcţiile `dlmread` şi `dlmwrite` amintite mai sus permit citirea şi scrierea unor valori separate prin delimitatori înt-un fişier de date ASCII. Un delimitator este orice caracter care separă valorile.

Exemplu:
Considerăm un fişier roger.dat ale cărui componente sunt separate prin punct şi virgulă:

```
7.2;8.5;6.2;6.6
5.4;9.2;8.1;7.2
```

Citirea şi transcrierea componentelor într-un tablou (matrice) A se face folosind ***dlmread:***

```
A = dlmread('roger.dat', ';');
A =
      1      2      3
      4      5      6
```

În mod similar se foloseşte ***dlmwrite*** pentru scrierea unui text cu delimitatori într-un fişier extern:

```
dlmwrite('roger',A,';')
```

roger va conţine:

```
1;2;3
4;5;6
```

Citirea fişierelor cu format uniform

Funcţia ***textread*** citeşte date de tip caracter sau numerice dintr-un fişier şi le transcrie în variabile MATLAB folosind specificatorii de conversie care definesc lungimea câmpului de date şi formatul acestora. Funcţia textread este utilă pentru fişiere cu format uniform şi cunoscut (de exemplu cu delimitatori de tip virgulă sau tab).

Exemplu: Fie fişierul **roger.dat** :

```
Salv    Type 25.74  65  Yes
```

Pentru citirea fişierului **roger.dat** ca fişier cu format liber se foloseşte formatul de conversie %:

```
[names,types,x,y,answer]=textread('roger.dat','%s %s %f %d
%s',1),
```

unde **%s** citeşte un şir de caractere separat prin spaţii albe, **%f** citeşte o valoare tip floating point, şi **%d** citeşte un întreg cu semn.
MATLAB® va răspunde:

```
names =
    'Salv'
types =
    'Type'
x =
  25.74000000000000
y =
    65
answer =
    'Yes'
```

3.4.4. Schimbarea de date între platforme (sisteme de operare)

În unele situaţii este necesar transferul de date şi programe între utilizatori care lucrează cu sisteme de operare diferite. Aplicaţiile MATLAB constau în fişiere .m cu funcţii şi script-uri şi fişiere tip .mat cu date binare. Ambele tipuri de fişiere pot fi transportate direct între diferite computere:

- Fişierele ***.m** conţin text simplu şi sunt independente de "maşină".
- Fişierele .mat sunt binare şi dependente de "maşină" dar pot fi transportate între computere deoarece conţin semnătura de "maşină" în antetul fişierului.

Observaţie Pentru utilizarea şi transportul aplicaţiilor MATLAB pe diverse computere (sisteme de operare) trebuie să ne asigurăm ***că fişierele .mat se transmit în binary file mode şi fişierele .m în ASCII file mode.***

Comanda diary
Comanda diary generează o copie a sesiunii de lucru MATLAB într-un fişier disc (fără grafice). Se poate vizualiza şi edita textul rezultat cu orice procesor de texte.
De exemplu, pentru crearea unui fişier cu numele **mai2013.out** care conţine comenzile şi ieşirile (răspunsurile) MATLAB corespunzătoare, trebuie tastat:
`diary mai2013.out`
Pentru oprirea înregistrării sesiunii se foloseşte:
`diary off`

3.5. Utilizarea memoriei

MATLAB® -ul necesită o zonă continuă de memorie pentru stocarea datelor din fiecare matrice. De asemenea, imaginile şi filmele (animaţia) cer o mare cantitate de memorie.
În plus, harta de pixeli (pixmap) folosită pentru imagini cere o cantitate de memorie proporţională cu suprafaţa imaginii de pe ecran. O imagine color de ***500x500 pixeli*** cere
1 Mb de memorie. Pentru limitarea memoriei necesare trebuie limitată dimensiunea imaginilor de pe ecran.

3.5.1. Rezolvarea erorilor de memorie

Dacă nu există memorie suficientă, în cazul unor matrici de dimensiuni mari este posibilă apariţia unei erori de memorie de tip "out of memory". Se poate încerca o defragmentare a memoriei cu comanda ***pack.***

Dacă astfel de erori "out of memory" sunt dese se pot încerca şi alte metode.



În cazul sistemelor Windows se creşte memoria virtuală folosind System Properties pentru Performance, instrument accesibil din Control Panel.



Pentru sisteme UNIX trebuie cerut administratorului de sistem să crească spaţiul ***swap*** [14].

CAPITOLUL 4.

PROGRAMAREA ÎN LIMBAJUL MATLAB

4.1. Fişiere MATLAB

Fişierele care conţin cod MATLAB sunt numite M-files sau fişiere *.m . După cum s-a precizat în capitolul de Fundamente ale programării în MATLAB, aceste fişiere pot fi funcţii (functions) care acceptă argumente de intrare şi furnizează ieşiri, sau pot fi fişiere script care execută o serie de instrucţiuni MATLAB. Pentru ca MATLAB-ul să recunoască un fişier ca fişier M-file trebuie ca numele acestuia să se termine cu extensia .m.
Fişierul * .m poate fi creat cu un editor de text şi apoi poate fi folosit ca orice funcţie sau comandă MATLAB:

1. Crearea unui fişier cu un editor de text.

```
function c = myfile(a,b)
c = sqrt((a.^2)+(b.^2))
```



2. Apelarea fişierului .m de la linia de comandă sau din alt fişier .m.

```
 a = 7.5
 b = 3.342
 c = myfile(a,b)

c =
```

Caracteristicile celor două tipuri de fişiere sunt prezentate în tabelul următor:

Fişiere Script	Fişiere Function
Nu accceptă argumente de intrare şi nu returnează ieşiri.	Acceptă argumente de intrare şi returnează ieşiri.
Operează cu datele din work-space.	Variabilele interne ale funcţiei sunt locale (implicit).
Utile pentru automatizarea unei serii de paşi care trebuie ex-ecutaţi de multe ori.	Utile pentru extinderea limbajului MATLAB pentru di-verse aplicaţii.

4.1.1. Script-uri

Fişierele script sunt cele mai simple fişiere MATLAB® nu au argumente de intrare sau de ieşire şi sunt utile pentru executarea secvenţială a unor calcule care altfel ar trebui executate

în mod repetat de la linia de comandă. Script-urile operează cu datele din workspace sau pot crea date noi. Aceste date sunt disponibile după terminarea execuţiei fişierului.

Părţile componente ale unui fişier de tip function

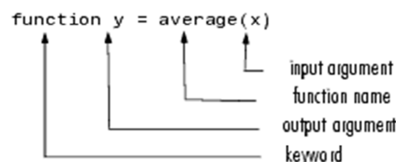
O funcţie .m are următoarele părţi componente:

- Linia de definire a funcţiei
- Linia de prim help H1
- Textul Help-ului
- Corpul funcţiei
- Comentarii

Linia de definire

Această linie informează MATLAB® -ul că fişierul conţine o funcţie şi specifică argumentele.

Exemplu:



Linia H1

Linia H1 este o linie de comentariu care începe cu semnul "%" şi furnizează prima linie text atunci când utilizatorul tastează help function_name la prompterul MATLAB®.

Textul Help-ului

Se poate crea un help online prin introducerea uneia sau mai multor linii de comentariu după linia H1, fiecare linie începând cu "%".

Corpul funcţiei

Corpul funcţiei conţine toate instrucţiunile în cod MATLAB® care permit efectuarea calculelor şi asignează valori argumentelor de ieşire. Declaraţiile din corp pot conţine apelări de funcţii, instrucţiuni de salt, intrări/ieşiri interactive, calcule etc.

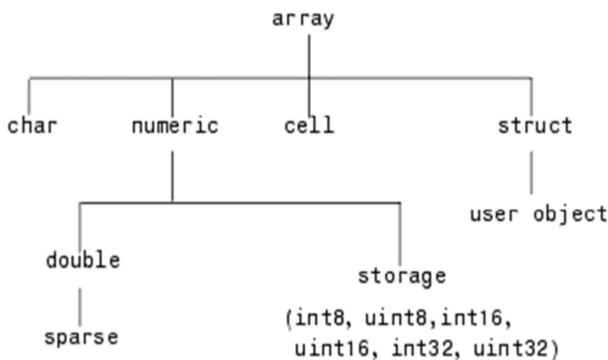
Comentarii

O linie de comentariu începe după cum s-a precizat cu semnul "%" şi poate fi plasată oriunde într-un fişier.
Pot fi adăugate linii albe oriunde în fişier, acestea fiind ignorate.

4.2. Tipuri de date şi operatori

4.2.1. Tipuri de date

MATLAB® -ul are şase tipuri fundamentale de date (sau clase), fiecare putând fi considerată ca tablou multidimensional. Cele şase clase sunt: `double, char, sparse, storage, cell şi struct`. Versiunile bi-dimensionale ale acestor tablouri sunt numite matrici şi de aici provine şi numele de MATLAB.



În tabelul următor sunt prezentate detaliat tipurile de date:

Clasa	Exemple	Descriere
Array		Tip de date virtual.
Cell	{17'hello'eye(2)}	Tablou tip celulă. Elementele celulei conţin alte tablouri.
Char	'Hello'	Tablou de tip caracter (sau şir de caractere–string); fiecare caracter are 16 biţi lungime.
Double	[1 2;3 4] 5+6I	Tablou numeric în dublă precizie (cel mai obişnuit tip de variabilă MATLAB).
Numeric		Tip de date virtual.
sparse	Speye(5)	Matrice de tip "sparse" în dublă precizie (doar 2-D). Tablourile de tip "sparse" stochează matrici cu doar câteva elemente nenule într-o fracţiune din spaţiul necesar unei matrici normale echivalente.
storage		Tip de date virtual.
struct	a.day = 12; a.color = 'Red'; a.mat = magic(3);	Tablou tip structură, care conţine nume de câmpuri, câmpuri care conţin alte tablouri.
uint8	Uint8(magic(3))	Tablou de numere întregi fără semn pe 8 biţi.
User Object	Inline('ln(x)')	Tip de date definit de utilizator.

4.2.2. Operatori

Operatorii MATLAB pot fi clasificaţi în trei categorii:

- Operatori aritmetici
- Operatori relaţionali care compară operanzii cantitativ
- Operatori logici

Operatori aritmetici:

+	Adunare	:	Operatorul două puncte
-	Scădere	.^	Putere
.*	Înmulţire	.'	Transpusa
./	Împărţire la dreapta	'	Transpusa complex conjugată
.\	Împărţire la stânga	*	Înmulţire de matrici
+	Plus unar	/	Împărţire matriceală la dreapta
-	Minus unar	\	Împărţire matriceală la stânga
		^	Putere de matrice

Cu excepţia unor operatori matriceali, operatorii aritmetici lucrează cu elementele corespondente ale unor tablouri de dimensiuni egale. Pentru vectori şi tablouri dreptunghiulare ambii operanzi trebuie să aibă aceeaşi dimensiune, cu excepţia situaţiei în care unul dintre ei este scalar. În acest caz MATLAB® -ul aplică scalarul fiecărui element al celuilalt operand (proprietatea de expansiune scalară).

Operatori relaţionali:

<	Mai mic
<=	Mai mic sau egal
>	Mai mare
>=	Mai mare sau egal
==	Egal cu
~=	Diferit de

Operatorii relaţionali compară elementele corespondente ale unor tablouri de dimensiune egală.
Operatorii relaţionali lucrează totdeauna element cu element.

Exemplu:

```
» A = [2 7 6;9 0 5;3 0.5 6];
» B = [8 7 0;3 2 5;4 -1 7];
» A == B
ans =
   0   1   0
   0   0   1
   0   0   0
```

Operatori logici:

&	AND (ŞI)
|	OR (SAU)
~	NOT (NU)

O expresie care utilizează operatorul & este adevărată dacă ambii operanzi sunt adevăraţi. În termeni numerici, expresia este adevărată dacă ambii operanzi sunt nenuli. Exemplu:

```
» u = [1 0 2 3 0 5];
» v = [5 6 1 0 0 7];
» u & v
ans =
   1   0   1   0   0   1
```

O expresie care utilizează operatorul | este adevărată dacă unul dintre operanzi este logic adevărat sau dacă ambii operanzi sunt adevăraţi. În termeni numerici, expresia este falsă dacă ambii operanzi sunt nuli.

Exemplu:

```
» u | v
ans =
   1   1   1   1   0   1
```

O expresie care utilizează operatorul NOT, ~, neagă operandul. În termeni numerici, orice operand nenul devine nul şi orice operand nul devine unu. Exemplu:

```
» ~u
ans =
     0     1     0     0     1     0
```

Operatorii logici lucrează cu elementele corespondente ale unor tablouri de dimensiuni egale. Pentru vectori şi tablouri dreptunghiulare ambii operanzi trebuie să aibă aceeaşi dimensiune, cu excepţia situaţiei în care unul dintre ei este scalar. În acest caz, ca şi la operatorii aritmetici, MATLAB® -ul aplică scalarul fiecărui element al celuilalt operand.

Funcţii logice

În plus faţă de operatorii logici MATLAB-ul furnizează şi funcţii logice:

Funcţie	Descriere	Exemple
xor	Realizează sau exclusiv. Returnează logic adevărat dacă unul din operanzi este adevărat şi celălalt fals. În termeni numerici, returnează 1 dacă un operand este nenul şi celălalt este zero.	» a = 1; » b = 1; » xor(a,b) ans = 0
all	Returnează 1 dacă toate elementele unui vector sunt adevărate sau nenule. Operează şi cu matrici (pe coloane).	» u = [0 1 2 0]; » all(u) ans = 0 » A = [0 1 2;3 5 0]; » all(A) ans = 0 1 0
any	Returnează 1 dacă oricare din elementele argumentului sunt adevărate sau nenule; în caz contrar returnează 0.	» v = [5 0 8]; » any(v) ans = 1

Alte funcţii: **`isnan, isinf, find`** (a se folosi help pentru detalii).

Prioritatea operatorilor

Deoarece se pot construi expresii cu diverse tipuri de operatori, nivelurile de prioritate determină ordinea în care sunt evaluate expresiile. În cadrul fiecărui nivel, operatorii au prioritate egală şi sunt evaluaţi de la stânga la dreapta.
Regulile de prioritate sunt prezentate în continuare, de la nivelul de prioritate cel mai mare spre cel mai mic.

Operator	Nivel de prioritate
()	Prioritate maximă
~ (negare)	
.' .^ ' ^ + (plus unar) - (minus unar)	
.* ./ .\ * / \	
+ (adunare) - (scădere)	
: < <= > >= == ~=	
& |	Prioritate minimă

4.3. Instrucţiuni de salt şi bucle

În MATLAB® există mai multe tipuri de instrucţiuni de control al buclelor:

- **`if`**, împreună cu else şi elseif execută un grup de instrucţiuni pe baza unei condiţii logice.
- **`switch, case şi otherwise`** execută diverse grupuri de instrucţiuni în funcţie de valoarea unei anumite condiţii logice.
- **`while`** execută un grup de instrucţiuni de un număr nedefinit de ori, pe baza unei condiţii logice.
- **`for`** execută un grup de instrucţiuni de un număr fixat de ori.
- **`break`** termină execuţia pentru o buclă `for` sau `while`.
- **`try...catch`** schimbă controlul buclei dacă o eroare este detectată în timpul execuţiei.
- **`return`** provoacă întoarcerea la funcţia care a apelat prcedura.

Toate instrucţiunile de salt folosesc comanda `end` pentru a indica sfârşitul blocului respectiv.
Exemple de utilizare a unor instrucţiuni de salt:

4.3.1. Instrucţiunile if şi elseif:

```
if n < 0 % Daca n este negativ afiseaza un mesaj de eroare.

disp('Intrarea trebuie sa fie pozitiva');

elseif rem(n,2)= =0  %Daca n este pozitiv si par, imparte-l la 2.

A = n/2;

else

A = (n+1)/2; %Daca n este pozitiv si impar incrementeaza şi împarte
la 2

end
```

4.3.2. Instrucţiunea for:

```
for i = 1:m
    for j = 1:n
        A(i,j) = 1/(i + j - 1);
    end
end
```

4.4. Evaluarea datelor de tip caracter

Evaluarea datelor de tip caracter asigură putere şi flexibilitate limbajului MATLAB.

4.4.1. Funcţiile eval şi feval

Funcţia eval evaluează un şir de caractere care conţine o expresie, o declaraţie sau un apel de funcţie. În cea mai simplă formă, sintaxa este următoarea:
`eval('string')`

Exemplu:
Evaluarea unei expresii folosite la generarea unei matrice Hilbert de ordinul `n`:

```
t = '1/(i+j-1)';
for i = 1:n
    for j = 1:n
        a(i,j) = eval(t);
    end
end
```

Alt exemplu de utilizare a funcţiei `eval` pentru o declaraţie:
`eval('t = clock')`

Funcţia feval diferă de eval prin faptul că execută o funcţie a cărui nume este într-un şir de caractere. Se poate folosi feval şi funcţia input pentru a alege din mai multe sarcini definite de fişiere .m .

Exemplu:

```
fun = ['sin'; 'cos'; 'log'];
k = input('Choose function number: ');
x = input('Enter value: ');
feval(fun(k,:),x)
```

Este indicată folosirea funcţiei `feval` în locul funcţiei `eval`, deoarece execuţia este mai rapidă.

4.4.2. Construirea şirurilor de caractere pentru evaluare

Se pot concatena şirurile de caractere pentru a crea expresii de intrare necesare funcţiei `eval`. În continuare este prezentat un exemplu în care funcţia eval creează 10 variabile numite `P1, P2, ...P10`, şi setează fiecare variabilă la o anumită valoare:

```
for i=1:10
    eval(['P',int2str(i),'= i.^2'])
end
```

4.5. Reprezentarea şi manipularea informaţiilor despre dată şi timp

MATLAB® -ul furnizează funcţii pentru manipularea informaţiilor despre dată şi timp, funcţii grupate în directorul ***timefun***.

Categorie	Funcţie	Descriere
Data şi timpul curent	now	Data şi timpul curent ca număr serial.
	date	Data curentă ca şir de caractere.
	clock	Data şi timpul curent ca vector.
Conversii	datenum	Conversia la număr serial al datei.
	datestr	Conversia la reprezentare de tip caracter.
	datevec	Componentele datei.
Utilitare	calendar	Calendar.
	weekday	Ziua din săptămână.
	eomday	Ultima zi din lună.
	datetick	Etichete formatate de tip dată.
Timing	cputime	Timpul CPU în secunde.
	tic, toc	Start şi oprire pentru timer.
	etime	Timp scurs.

4.6. Intrări utilizator

Pentru a obţine o intrare de la utilizator în timpul execuţiei unui fişier există următoarele posibilităţi:
Afişarea unui prompter prin intermediul unei funcţii tip input şi introducerea unor date de la tastatură.
Oprirea execuţiei cu o comandă pause (reluarea execuţiei la apăsarea unei taste).
Construirea unei interfeţe grafice GUI completă [14] .

Funcţia input asigură afişarea unui prompter şi aşteaptă un răspuns de la utilizator.
Sintaxa este:

```
n = input('prompt_string')
```

Funcţia determină afişarea şirului de caractere prompt_string, aşteaptă o intrare de la tastatură şi returnează valoarea introdusă de la tastatură. Funcţia este utilă pentru implementarea aplicaţiilor de tip meniu. Această funcţie poate să returneze intrarea de la utilizator sub formă de caracter.
Exemplu:

```
name = input('Enter address: ','s');
```

Comanda pause, fără argumente, opreşte execuţia până la apăsarea unei taste. Pentru a avea o pauză de n secunde se foloseşte comanda:

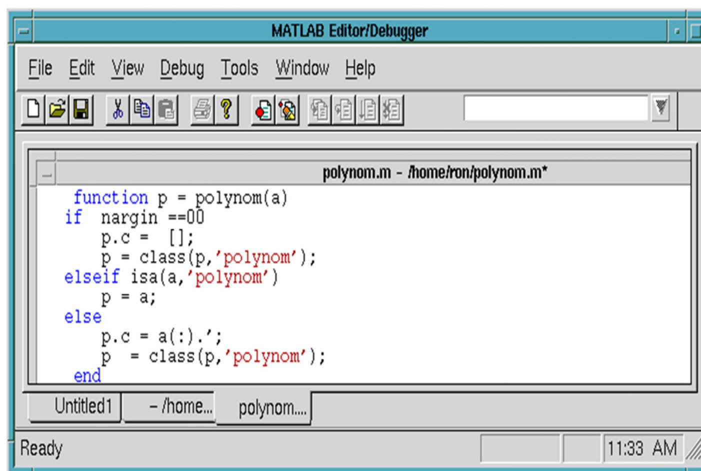
```
pause(n)
```

CAPITOLUL 5.

EDITORUL/DEBUGGER-UL ŞI PROFILER-UL MATLAB®

5.1. Editorul/Debugger-ul MATLAB®

MATLAB® -ul dispune de un editor propriu, editor care este asociat şi cu un program de depanare (debugger). Editorul/debugger-ul oferă posibilitatea de a efectua operaţiunile de editare de bază precum şi accesul la instrumente de depanare a fişierelor .m .
Pachetul Editor/Debugger [16] oferă şi o interfaţă grafică uşor de utilizat. Pentru lansarea editorului se tastează la prompterul MATLAB comanda ***edit***. Se pot folosi şi butoanele/meniurile accesibile din fereastra de comandă.



5.1.1. Setarea implicită a Editorului

Facilităţile de editare şi depanare sunt setate să fie active în mod implicit atunci când MATLAB-ul este instalat.
Dacă se doreşte instalarea altui editor sau nu se doreşte apelarea la depanarea în regim grafic se pot dezactiva aceste facilităţi prin setarea corespunzătoare.
De exemplu (în UNIX) se modifică ~home/.Xdefaults file:

```
matlab*builtInEditor: Off
matlab*graphicalDebugger: Off
```

Trebuie rulat

```
xrdb -merge ~home/.Xdefaults
```

`înainte de pornirea MATLAB-ului.`

5.1.2. Debugger-ul *MATLAB®*.

Exemple de depanare a fişierelor MATLAB®

Debugger-ul permite identificarea erorilor de programare. Prin folosirea debugger-ului se poate vizualiza conţinutul workspace-ului în orice moment în timpul execuţiei unei funcţii şi se poate executa programul (codul) MATLAB® linie cu linie. Pentru folosirea acestui instrument de depanare se poate utiliza interfaţa grafică a debugger-ului sau se pot folosi linii de comandă.

Depanarea permite corectarea a două tipuri de erori:

- Erori de sintaxă, cum ar fi scrierea incorectă a numelui unei funcţii sau omiterea unor paranteze. MATLAB® -ul detectează majoritatea acestor erori şi afişează un mesaj de eroare care descrie natura erorii şi numărul liniei din programul .m în care a apărut eroarea respectivă.
- Erori de rulare (de calcul), care sunt mai mult de natură algoritmică. De exemplu este posibil să modificăm o altă variabilă decât trebuie sau să efectuăm un calcul incorect. Aceste erori sunt observate atunci când fişierul rulat furnizează rezultate necorespunzătoare.

În timp ce erorile de sintaxă se corectează relativ uşor pe baza mesajelor de eroare, erorile de rulare sunt mai greu de depanat. Se pot utiliza în acest caz mai multe tehnici de depanare. Se îndepărtează delimitatorii de tip punct şi virgulă de la sfârşitul liniilor program. Astfel la rularea programului vor fi afişate şi rezultatele intermediare corespunzătoare fiecărei linii.

Se adaugă comanda keyboard în fişierele .m care sunt depanate. Această comandă opreşte execuţia programului respectiv, dă controlul la tastatură şi dă posibilitatea examinării şi schimbării unor funcţii sau variabile. Acest mod de lucru este indicat printr-un prompter special:

`"K>>."`

Pentru a continua execuţia, se tastează return şi se apasă apoi tasta Return şi
se folosesc în continuare comenzi ale Debugger-ul MATLAB® .

Debugger-ul este util deoarece permite accesul la funcţiile din workspace, examinarea şi eventual modificarea conţinutului acestora.

Debugger-ul permite setarea sau ştergerea unor puncte de oprire: breakpoints, care sunt linii speciale în programul MATLAB® la întâlnirea cărora execuţia se opreşte şi sunt posibile operaţii de schimbare şi de execuţie a liniilor de comandă una câte una.

Exemplu de depanare:

Pentru a ilustra procedurile de depanare disponibile (îndeosebi pentru cazul erorilor de rulare) vom folosi un exemplu preluat din MATLAB® User Guide [16]. Vom scrie un fişier denumit variance.m care este o funcţie având ca intrare un vector şi ca ieşire un scalar. Fişierul apelează la o altă funcţie, numită sqsum, care calculează o sumă medie pătratică a vectorului de intrare.

```
function y = variance(x)
mu = sum(x)/length(x);
tot = sqsum(x,mu);
y = tot/(length(x)-1);
```

În fişierul **sqsum.m** se strecoară intenţionat o eroare:

```
function tot = sqsum(x,mu)
tot = 0;
for i = 1:length(mu)
    tot = tot + ((x(i)-mu).^2);
end
```

Pentru verificarea corectitudinii calculelor, folosim funcţia MATLAB® **`std`** (calculează deviaţia standard) care permite efectuarea unui calcul echivalent.
Se introduce mai întâi un vector de intrare de test:

```
» v = [1 2 3 4 5];
```

apoi se utilizează funcţia std:

```
» var1 = std(v).^2
var1 =
    2.5000
```

Încercăm funcţia variance care apelează funcţia **sqsum** (scrisă eronat):

```
» myvar1 = variance(v)
myvar1 =
     1
```

Răspunsul este greşit. Vom încerca cu debugger-ul să găsim şi să corectăm greşeala.

5.1.3. Depanarea cu ajutorul interfeţei grafice a Debugger-ului

A. Pentru startarea procedurii de depanare

Dacă ***fişierul .m (adică variance.m)*** a fost creat cu editorul MATLAB şi suntem în fereastra Editor/Debugger, se continuă din acest punct. Dacă fişierul a fost creat cu un editor extern, se startează Editor/Debugger-ul şi apoi se face click pe butonul Open M-file din toolbar. ***Toolbar-ul Editor/Debugger*** conţine o serie butoane descrise în continuare:

Buton Toolbar	Scop	Descriere	Comandă Echivalentă
	Setează/ Şterge Breakpoint	Setează sau şterge un breakpoint pe linia pe care este poziţionat cursorul.	Dbstop/ Dbclear
	Şterge toate Break-point-urile	Şterge toate breakpoint-urile care sunt setate în mod curent.	Dbclear all
	Step In (Pas în)	Execută linia curentă a fişierului .m şi dacă linia este o apelare la altă funcţie sare (face un pas) în funcţia respectivă.	Dbstep in
	Single Step (Un singur pas)	Execută linia curentă a fişierului .m .	Dbstep
	Continuă	Continuă execuţia fişierului până la terminare sau până la alt breakpoint.	Dbcont
	Sfârşit depanare	Ieşirea din starea de depanare.	dbquit

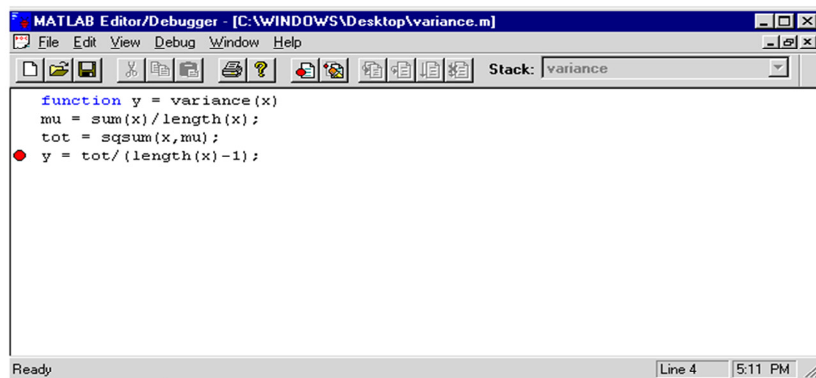
Prin apăsarea butonului din dreapta al mouse-ului în fereastra editorului se poate obţine un meniu cu toate aceste opţiuni.

B. Setarea Breakpoint-urilor

Punctele de oprire (breakpoint-uri) determină oprirea execuţiei fişierului la linia specificată şi permit evaluarea şi schimbarea variabilelor din workspace înainte de reluarea execuţiei. Breakpoint-ul este indicat printr-un semn roşu de stop (●) înainte de linia respectivă.

Pentru exemplul considerat, la începutul depanării nu se ştie unde ar putea fi eroarea, însă un loc logic de amplasare a unui breakpoint pentru a face verificări este în linia 4 a funcţiei variance.m:

```
y = tot/(length(x)-1);
```



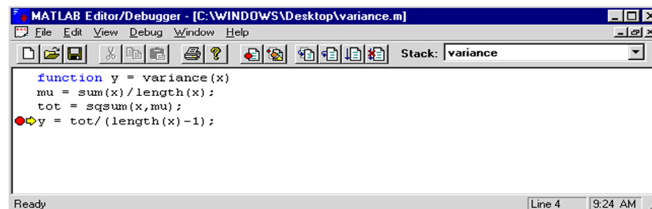
Pentru setarea breakpointului se poziţionează cursorul pe linia 4 şi se face click pe butonul din toolbar sau se alege Set Breakpoint din meniul **Debug.**

C. Examinarea variabilelor

Pentru verificarea variabilelor, se execută mai întâi funcţia din fereastra de comandă:

```
variance(v)
```

Atunci când execuţia programului ajunge la breakpoint, o săgeată galbenă orizontală (=>) arată următoarea linie care va fi executată. Dacă săgeata galbenă este verticală , atunci aceasta indică o pauză la sfârşitul unui script sau a unei funcţii şi permite examinarea variabilelor înainte de reîntoarcerea la funcţia principală.



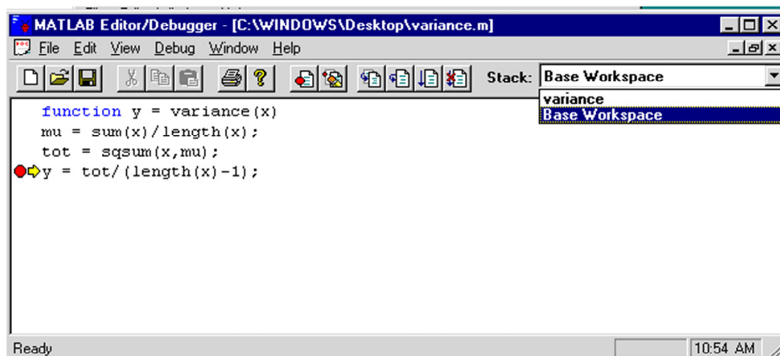
Acum putem verifica valorile variabilelor mu şi tot. Se selectează textul care conţine variabilele şi se face click din butonul drept al mouse-ului după care se alege din meniu Evaluate Selection.
În fereastra de comandă va fi afişat atât textul selectat cât şi rezultatul:

```
K>> mu
mu =
      3
K>> tot
tot =
      4
```

Din analiza acestor valori se observă că eroarea se află în **sqsum**.

D. Schimbarea contextului spaţiului de lucru

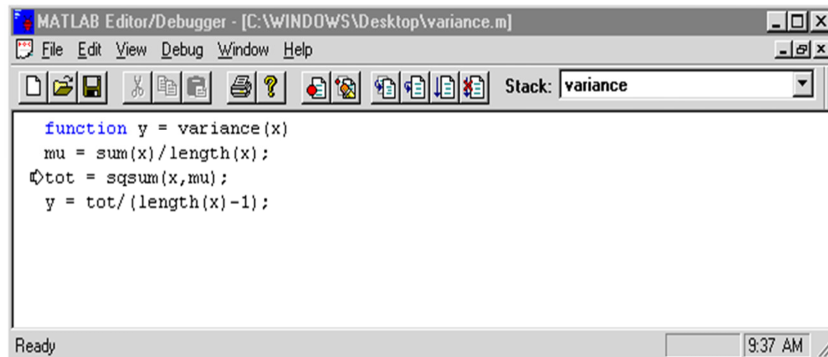
Se poate folosi meniul Stack pentru schimbarea contextului spaţiului de lucru, adică pentru ieşirea din funcţia variance şi vizualizarea conţinutului workspace-ului, prin selectarea din meniu a opţiunii Base Workspace:



Prin utilizarea comenzii whos sau a Browserului Workspace se vor vizualiza variabilele v şi myvar1, ca de altfel şi celelalte variabile create. Pentru întoarcerea la contextul spaţiului de lucru local al funcţiei variance se selectează **Variance** din meniu.

E. Executarea pas cu pas a programului şi continuarea execuţiei

Se şterge breakpoint-ul din linia 4 din variance.m prin plasarea cursorului pe linie şi selectarea opţiunii `Clear Breakpoint` din meniul Debug. Se continuă execuţia programului cu `Continue` din meniul `Debug`.
Se deschide sqsum.m şi se setează un breakpoint la linia 4 pentru verificarea buclei şi a calculelor. Se rulează din nou `variance`.
Execuţia se va opri acum la linia 4 din `sqsum`.



Se poate acum evalua indicele buclei i:

```
K>> i
i =
      1
```

după care prin selectarea opţiunii Single Step din meniul Debug se execută linia următoare. Se evaluează variabila tot:

```
K>> tot
tot =
       4
```

Se selectează din nou Single Step:

```
for i  = 1:length(mu)
```

Se observă că bucla este iterată numai până la lungimea lui mu, care este scalar, şi nu până la lungimea lui x, vectorul de intrare (aceasta este de fapt greşeala).
O dată eroarea găsită se selectează **Quit Debugging** şi se termină execuţia programului. Se şterge breakpoint din sqsum şi se pune un breakpoint la linia 4 din variance.m, după care rulăm din nou:

```
variance(v)
```

La oprirea execuţiei se setează valoarea lui tot la valoarea corectă 10:

```
K>> tot = 10
tot =
        10
```

Selectăm ***Continue Execution*** şi obţinem rezultatul corect.

F. Terminarea sesiunii de depanare

Se selectează Exit Editor/Debugger din meniul File şi se termină sesiunea de depanare.
Pentru corectarea definitivă a erorii se foloseşte editorul şi se rulează din nou programul pentru o ultimă verificare.

G. Depanarea în linia de comandă

Folosirea facilităţilor de depanare se poate realiza şi direct din linia de comandă, prin intermediul unui set de comenzi. Aceste comenzi sunt prezentate în forma lor generală în tabelul următor:

Descriere	Sintaxă
Setarea unui breakpoint.	dbstop at line_num in file_name
Ştergerea unui breakpoint.	dbclear at line_num in file_name
Stop la atenţionare, eroare sau generarea de NaN/Inf.	dstop if warning error naninf infnan
Reluarea execuţiei.	Dbcont
Listarea apelării de funcţii.	Dbstack
Listarea tuturor breakpoint-urilor.	dbstatus file_name
Executarea a una sau mai multe linii. Descriere	Dbstep nlines Sintaxă
Listarea fişierelor M-file cu liniile numerotate.	dbtype file_name
Schimbarea contextului spaţiului de lucru local (down).	dbdown
Schimbarea contextului spaţiului de lucru local (up).	dbup
Părăsirea modului de depanare.	dbquit

Pentru informaţii suplimentare privind utilizarea acestor funcţii se poate apela la comanda `help` urmată de numele comenzii respective.
Observaţie: Exemplul de depanare a unui fişier cu erori prezentat anterior poate fi reluat, utilizându-se în locul interfeţei grafice a debugger-ului comenzi corespunzătoare în linia de comandă.

5.2. Profiler-ul MATLAB®

Pentru îmbunătăţirea performanţelor fişierelor MATLAB® se utilizează un instrument MATLAB® numit ***Profiler***. Acest instrument furnizează informaţii utile privitoare la timpul alocat calculelor efectuate de fiecare linie program.
Cu ajutorul ***Profiler***-ului se măsoară modul în care programul consumă timp, şi o măsură este evident mai bună decât ghicitul rutinelor sau funcţiilor care consumă mult timp de cal-cul.
Programarea eficientă presupune folosirea ***Profiler***-ului pentru determinarea "strangulărilor" din programul creat şi apoi modificarea programului pentru optimizarea timpului de calcul.

Programele MATLAB® au în general o structură multistrat generată de faptul că funcţiile utilizate apelează deseori alte funcţii şi aşa mai departe. De aceea este important să fie identificate acele funcţii consumatoare de timp şi înlocuite dacă este posibil.
Profiler-ul permite:

- Evitarea calculelor inutile.
- Schimbarea algoritmilor pentru evitarea folosirii unor funcţii consumatoare de timp.
- Evitarea recalculărilor prin stocarea unor rezultate ce pot fi utilizate ulterior.

5.2.1. Comanda profile

Pentru a crea un profil al programului (fişierului) MATLAB® se foloseşte comanda ***profile*** pentru a genera şi vizualiza statisticile despre programul respectiv. În tabelul următor sunt prezentate formele posibile ale acestei comenzi.

Sintaxă	Opţiuni	Descriere
Profile on		Lansează profiler-ul şi şterge statisticile înregistrate anterior.
	-detail level	Specifică nivelul funcţiei analizate.
	-history	Specifică secvenţa exactă a apelurilor făcute de funcţia care va fi înregistrată.
Profile report		Suspendă activitatea profilerului după care generează un raport în format HTML pe care îl afişează în browserul Web.
	Base-name	Salvează raportul în fişierul basename din directorul curent.
Profile plot		Suspendă activitatea profiler-ului după care afişează un grafic în fereastra figură cu funcţiile care consumă majoritatea timpului de execuţie.
Profile resume		Restartează profiler-ul fără a şterge statisticile înregistrate anterior.
Profile clear		Şterge statisticile înregistrate.
Profile off		Termină activitatea profiler-ului.
Profile status		Afişează o structură care conţine starea curentă a profiler-ului.
stats = pro-file('info')		Suspendă profiler-ul şi afişează o structură cu rezultatele activităţii de analiză.

Exemplu de utilizare a Profiler-ului:
Se startează profiler-ul:

```
profile on -detail builtin -history
```

Opţiunea -detail builtin determină profilerul să întocmească statistici şi pentru funcţiile built-in.
Se execută un fişier .m .
În exemplu următor este preluat programul care rulează modelul Lotka-Volterra pentru populaţii tip prădător-pradă (lotkademo pentru demo).
```
[t,y] = ode45('lotka',[0 2],[20;20]);
```
Se generează un raport şi se salvează rezultatele în fişierul lotkaprof.
```
profile report lotkaprof
```
Se restartează profiler-ul fără ştergerea statisticilor existente.
```
profile resume
```
Se opreşte profiler-ul.
```
profile off
```

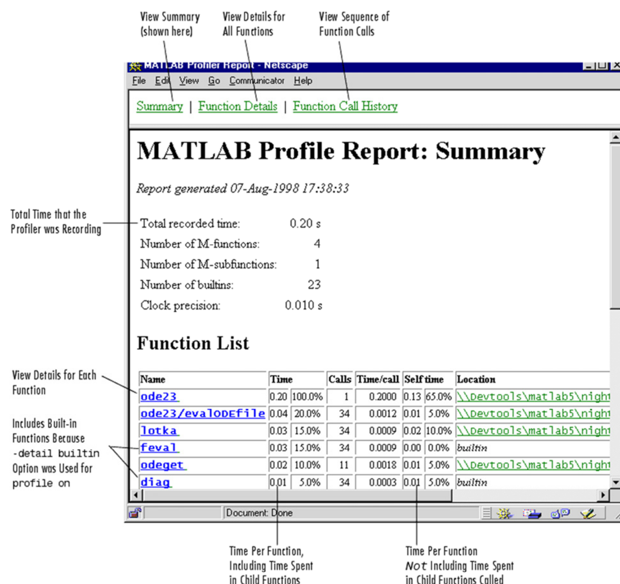
5.2.2. Vizualizarea rezultatelor

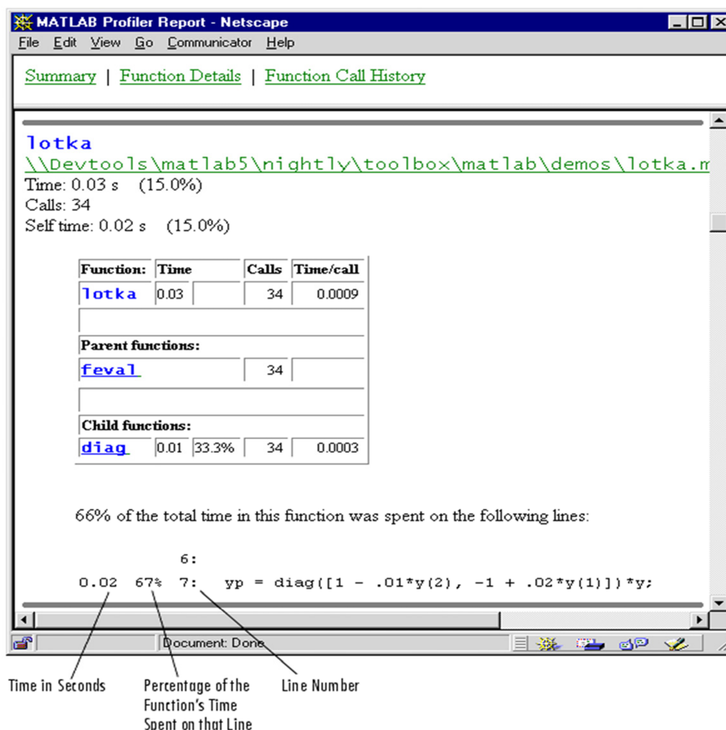
A. Rapoarte

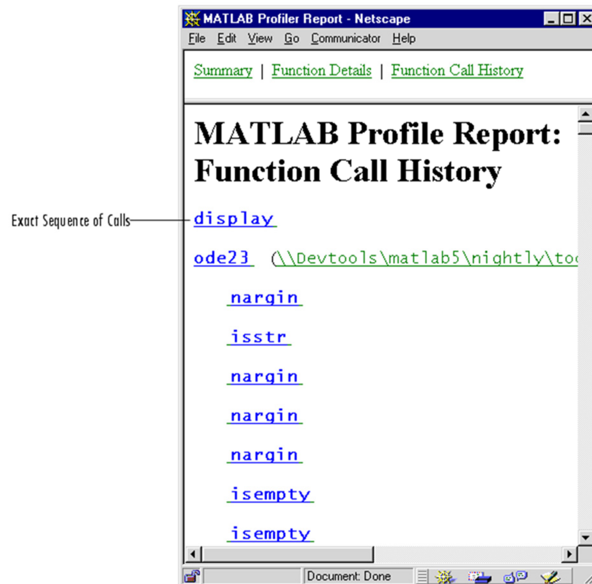
Pentru afişarea unui raport cu rezultatele statistice obţinute se tastează profile report
Raportul care rezultă apare în fereastra browserului Web şi începe cu un rezumat al raportului din care se pot accesa un raport detaliat şi un raport al apelărilor de funcţii (o cronică).

- Raportul rezumat. În figura următoare este prezentat raporul rezumat pentru ***exemplul Lotka-Volterra*** [20].
- Raportul detaliat. Acest raport furnizează detalii despre funcţiile de tip "părinte" şi "copil" ale unei funcţii. Este prezentat raportul detaliat pentru funcţia lotka din exemplul considerat.
- Raportul apelărilor de funcţii. Acest raport afişează secvenţa exactă a funcţiilor apelate. Pentru a vizualiza acest raport, trebuie startat profiler-ul cu opţiunea -history.

```
profile on -history
```
Este prezentat un exemplu de astfel de raport.


View Summary (shown here)
View Details for All Functions
View Sequence of Function Calls
MATLAB Profiler Report - Netscape
File Edit View Go Communicator Help
Summary | Function Details | Function Call History
MATLAB Profile Report: Summary
Report generated 07-Aug-1998 17:38:33
Total Time that the Profiler was Recording
Total recorded time: 0.20 s
Number of M-functions: 4
Number of M-subfunctions: 1
Number of builtins: 23
Clock precision: 0.010 s
Function List
View Details for Each Function
Includes Built-in Functions Because -detail builtin Option was Used for profile on
Name | Time | Calls | Time/call | Self time | Location
ode23 | 0.20 | 100.0% | 1 | 0.2000 | 0.13 | 65.0% | \\Devtools\matlab5\night
ode23/evalODEfile | 0.04 | 20.0% | 34 | 0.0012 | 0.01 | 5.0% | \\Devtools\matlab5\night
lotka | 0.03 | 15.0% | 34 | 0.0009 | 0.02 | 10.0% | \\Devtools\matlab5\night
feval | 0.03 | 15.0% | 34 | 0.0009 | 0.00 | 0.0% | builtin
odeget | 0.02 | 10.0% | 11 | 0.0018 | 0.01 | 5.0% | \\Devtools\matlab5\night
diag | 0.01 | 5.0% | 34 | 0.0003 | 0.01 | 5.0% | builtin
Document: Done
Time Per Function, Including Time Spent in Child Functions
Time Per Function Not Including Time Spent in Child Functions Called


MATLAB Profiler Report - Netscape
File Edit View Go Communicator Help
Summary | Function Details | Function Call History
lotka
\\Devtools\matlab5\nightly\toolbox\matlab\demos\lotka.m
Time: 0.03 s (15.0%)
Calls: 34
Self time: 0.02 s (15.0%)
Function: | Time | Calls | Time/call
lotka | 0.03 | 34 | 0.0009
Parent functions:
feval | 34
Child functions:
diag | 0.01 | 33.3% | 34 | 0.0003
66% of the total time in this function was spent on the following lines:
6:
0.02 67% 7: yp = diag([1 - .01*y(2), -1 + .02*y(1)])*y;
Document: Done
Time in Seconds
Percentage of the Function's Time Spent on that Line
Line Number

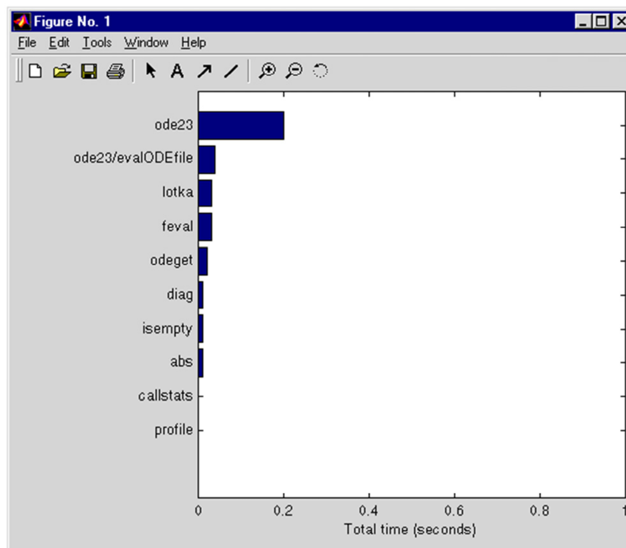


B. Reprezentarea grafică a rezultatelor Profiler-ului

Pentru a obţine o reprezentare grafică trebuie să tastăm în linia de comandă:

```
>> profile plot.
```

În fereastra grafică va apare un grafic de forma din figura următoare:



CAPITOLUL 6.

GRAFICE ŞI INTERFEŢE GRAFICE ÎN MATLAB

6.1. Tehnici de reprezentare grafică

În general, pentru a realiza o reprezentare grafică, trebuie parcurse etapele următoare [13] :

Etapa	Instrucţiuni
1.Pregătirea datelor	x = 0:0.2:12; y1 = bessel(1,x); y2 = bessel(2,x); y3 = bessel(3,x);
2. Selectarea ferestrei grafice şi poziţonarea graficului în fereastră	figure(1) subplot(2,2,1)
3. Apelarea unei funcţii elementare de plotare	h = plot(x,y1,x,y2,x,y3);
4. Selectarea caracteristici-lor liniei şi markerului.	set(h,'LineWidth',2,{'LineStyle'},{'--';':';'-.'}) set(h,{'Color'},{'r';'g';'b'})
5. Setarea limitelor axelor, gridare (caroiere)	axis([0 12 -0.5 1]) grid on
6. Completarea graficului cu etichete pe axe, legendă, text	xlabel('Time') ylabel('Amplitude') legend(h,'First','Second','Third') title('Bessel Functions') [y,ix] = min(y1); text(x(ix),y,'First Min \rightarrow',... 'HorizontalAlignment','right')
7. Export grafice	print -depsc -tiff -r200 myplot

Funcţiile de bază folosite la reprezentarea grafică sunt prezentate în tabelul următor:

Funcţie	Utilizare
Plot	Generează grafice 2-D cu scalare liniară a axelor
Plot3	Generează grafice 3-D cu scalare liniară a axelor
loglog	Generează grafice cu scalare logaritmică a axelor
semilogx	Generează grafice cu scalare liniară a axei y şi cu scalare logaritmică a axei x
semilogy	Generează grafice cu scalare liniară a axei x şi cu scalare logaritmică a axei y
plotyy	Generează grafice cu dublă reprezentare a axei y (pe stânga şi pe dreapta)

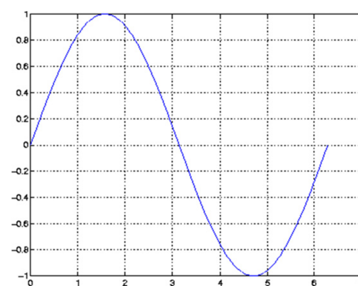
6.1.1. Reprezentări grafice 2-D

Generarea graficelor

Funcţia `plot` are diferite forme în funcţie de argumentele de intrare.
Dacă de exemplu y este un vector, `plot(y)` produce un grafic liniar al elementelor lui y versus indexul elementelor sale.
Dacă se specifică doi vectori ca argumente, `plot(x,y)` produce graficul lui y versus x.

Exemplu:

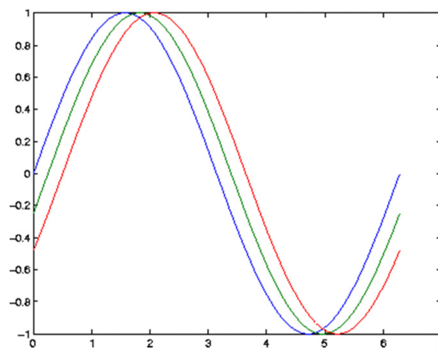
```
t = 0:pi/100:2*pi;
y = sin(t);
plot(t,y);grid
```



Se pot realiza grafice multiple utilizând un singur apel al funcţiei `plot` MATLAB® -ul realizează automat o reprezentare cu culori diferite pentru a permite distingerea graficelor.

Exemplu:

```
y2 = sin(t-0.25);
y3 = sin(t-0.5);
plot(t,y,t,y2,t,y3)
```

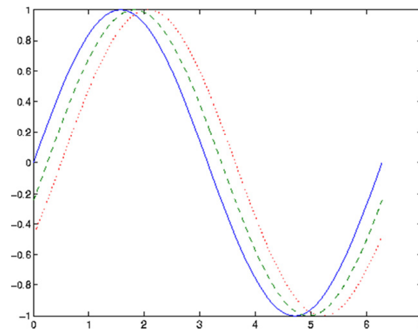


Specificarea stilului de linie

Se pot crea diferite tipuri de linii pentru fiecare set de date prin folosirea unor identificatori de tip `string` în funcţia `plot`.

Exemplu:

```
t = 0:pi/100:2*pi;
y = sin(t);
y2 = sin(t-0.25);
y3 = sin(t-0.5);
plot(t,y,'-',t,y2,'--',t,y3,':')
```



Funcţiile de plotare acceptă deci argumente de tip caracter care specifică stilul liniei, simbolurile utilizate pentru marker, culoarea etc. Forma generală este:

`plot(x,y,'linestyle_marker_color'),`

unde linestyle_marker_color este un şir de caractere construit din:

- Un stil de linie (de exemplu linie punctată, plină etc.)
- Un tip de marker (de exemplu x, *, o, etc.)
- Un specificator de culoare (c, m, y, k, r, g, b, w)

Se poate folosi un specificator sau mai mulţi, în orice ordine.

De **exemplu,**

`'go--' %` defineşte o linie întreruptă, cu markere circulare, ambele colorate în verde.

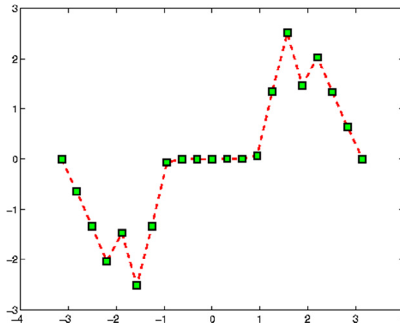
Specificarea culorii şi dimensiunii liniilor

Caracteristicile liniilor se pot controla prin specificarea unor valori pentru proprietăţile linilor:

- **`LineWidth`** – specifică lăţimea unei linii.
- **`MarkerEdgeColor`** – setează culoarea markerului sau culoarea marginilor markerului în cazul anumitor forme (cerc, pătrat etc.)
- **`MarkerFaceColor`** – setează culoarea interiorului markerelor.
- **`MarkerSize`** – specifică dimensiunea markerului.

Exemplu:

```
x = -pi:pi/10:pi;
y = tan(sin(x)) - sin(tan(x));
plot(x,y,'--rs','LineWidth',2,...
                'MarkerEdgeColor','k',...
                'MarkerFaceColor','g',...
                'MarkerSize',10)
```



Suprapunerea unor grafice peste un grafic existent

Se pot adăuga grafice peste unul existent cu comanda hold. Dacă se setează hold on, MATLAB® -ul nu înlătură graficul existent, ci suprapune noul grafic în aceeaşi fereastră grafică.

Exemplu:

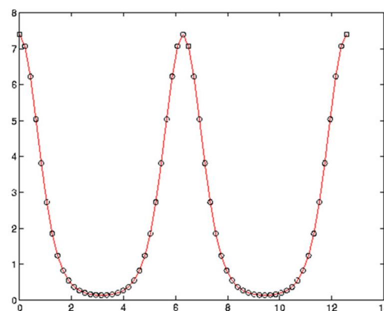
```
semilogx(1:100,'+')
hold on
plot(1:3:300,1:100,'--')
hold off
```

Reprezentarea grafică simultană a markerelor şi liniilor

Pentru reprezentarea grafică a markerelor (care indică punctele corespunzătoare datelor) şi a liniilor (care unesc aceste date) se specifică atât tipul markerului cât şi stilul liniei.

Exemplu:

```
x = 0:pi/15:4*pi;
y = exp(2*cos(x));
plot(x,y,'-r',x,y,'ok')
```



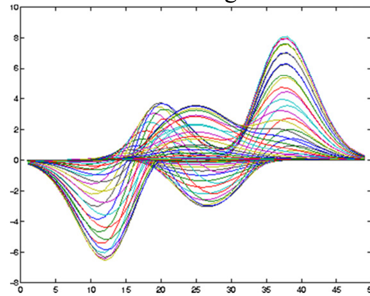
Reprezentarea grafică a datelor din matrici

Atunci când funcţia plot este utilizată cu un singur argument de tip matrice:

```
plot(y)
```

va fi realizat un grafic pentru fiecare coloană a matricii, cu axa x reprezentând indexul de linie 1:m, cu m numărul liniilor din Y.

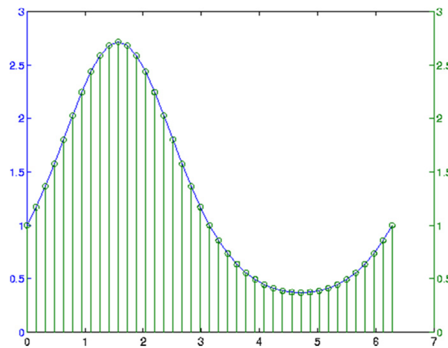
Exemplu:
Cu instrucţiunea `Z = peaks;`
este creată o matrice ***49 x 49*** obţinută printr-o evaluare de funcţie.
Dacă plotăm matricea cu `plot(Z)` vom avea un grafic cu 49 de linii.



Reprezentarea grafică cu axa Y dublă

Comanda `plotyy` permite crearea unor grafice pentru două seturi de date şi cu reprezentare dublă a axei Y, pe partea stângă şi pe partea dreaptă.

Exemplu
```
t = 0:pi/20:2*pi;
y = exp(sin(t));
plotyy(t,y,t,y,'plot','stem')
```



Setarea parametrilor axelor

MATLAB® -ul setează automat limitele axelor şi gradarea acestora. Se pot însă folosi şi setările utilizatorului, cu comenzile:
`axis` – setează axele pentru fereastra grafică curentă.
`axes` – creează axe noi cu caracteristici specificate.
`get şi set` – permit obţinerea şi setarea unor proprietăţi ale axelor.
`gca` – returnează identificatorul axelor curente.
Se pot parcurge în detaliu aceste comenzi prin apelarea la help.

Ferestre de tip figură

MATLAB® -ul direcţionează ieşirile grafice spre o fereastră distinctă de fereastra de comandă. Această fereastră grafică este denumită figură (figure). (a se vedea paragraful 3.3).

Funcţia figure generează ferestre grafice. De exemplu,

```
>> figure ( )% generează o nouă fereastră şi o face fereastra curentă.
```

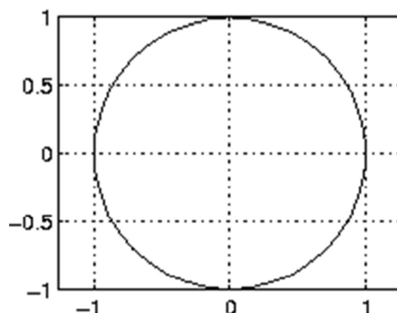
Afişarea unor în aceeaşi fereastră grafică a mai multor grafice

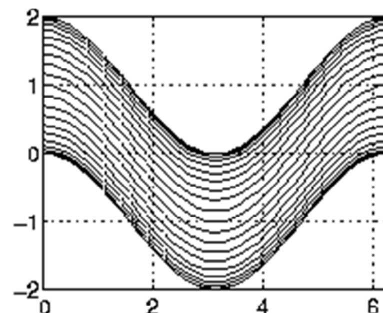
Se poate realiza o afişare a mai multor grafice în aceeaşi fereastră prin intermediul funcţiei `subplot`.

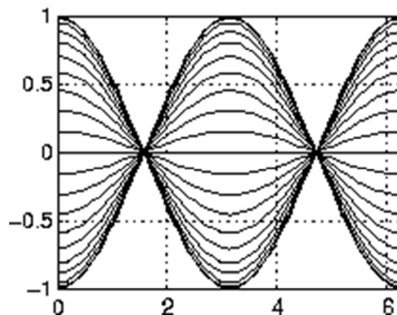
Funcţia `subplot(m,n,i)` desparte fereastra de tip figură într-o matrice m x n de mici subgrafice şi selectează subgraficul i ca grafic curent.

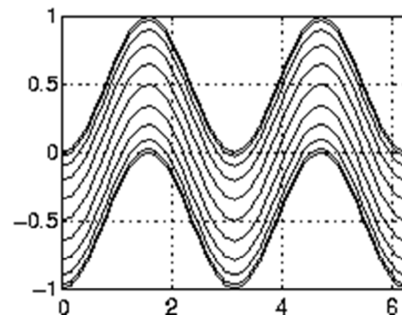
Exemplu:

```
t = 0:pi/20:2*pi;
[x,y] = meshgrid(t);
subplot(2,2,1)
plot(sin(t),cos(t))
axis equal
subplot(2,2,2)
z = sin(x)+cos(y);
plot(t,z)
axis([0 2*pi -2 2])
subplot(2,2,3)
z = sin(x).*cos(y);
plot(t,z)
axis([0 2*pi -1 1])
subplot(2,2,4)
z = (sin(x).^2)-(cos(y).^2);
plot(t,z)
axis([0 2*pi -1 1])
```









Comenzi de marcare, etichetare şi gradare a graficelor

MATLAB® -ul furnizează comenzi de etichetare a fiecărei axe şi de plasare a unui text în orice loc din grafic.
Comenzile sunt prezentate în tabelul următor:

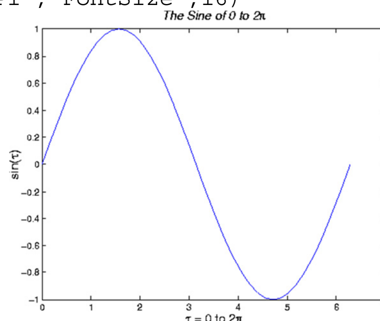
Comandă	Descriere
title	Adaugă un titlu
xlabel	Adaugă o etichetă pe axa x
ylabel	Adaugă o etichetă pe axa y
zlabel	Adaugă o etichetă pe axa z
legend	Adaugă o legendă
Text	Afişează un text la o locaţie specificată
Gtext	Plasează textul pe grafic utilizând mouse-ul

Etichetarea axelor

Se pot adăuga etichete pe axe cu comenzile : ***xlabel, ylabel, zlabel.***

Exemplu:

```
xlabel('t = 0 to 2\pi','FontSize',16)
ylabel('sin(t)','FontSize',16)
title('Value of the Sine from Zero to Two
                Pi','FontSize',16)
```



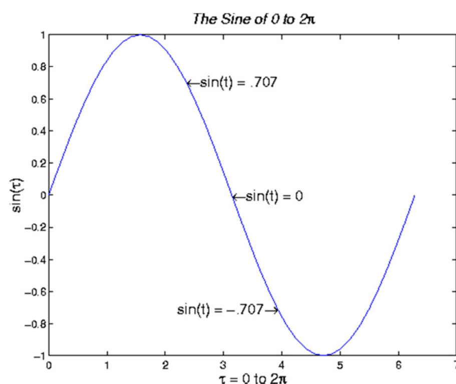
MATLAB® -ul interpretează caracterele care urmează după backslash "\" ca şi comenzi `LaTeX`. Aceste comenzi permit reprezentarea unor simboluri cum ar fi literele greceşti sau săgeţile.

Adăugarea textelor

Prin utilizarea funcţiei text se poate plasa un text (şir de caractere) oriunde pe grafic.

Exemplu:

```
text(3*pi/4,sin(3*pi/4),...
      '\leftarrowsin(t) = .707',...
      'FontSize',16)
text(pi,sin(pi),'\leftarrowsin(t) = 0',...
      'FontSize',16)
text(5*pi/4,sin(5*pi/4),'sin(t)=-.707\rightarrow',...
      'HorizontalAlignment','right',...
      'FontSize',16)
```



Plasarea textului în mod interactiv

Dacă utilizăm funcţia **`gtext`** se poate plasa un text în mod interactiv, cu mouse-ul, oriunde pe grafic. Această funcţie acceptă ca argument un şir de caractere şi aşteaptă până când utilizatorul selectează un loc pe grafic cu ajutorul mouse-ului.
Se poate utiliza şi `Plot Editor` pentru plasarea textului.

6.1.2. Grafice 2 D specializate

MATLAB® -ul permite lucrul cu o mare varietate de tipuri de grafice, astfel încât informaţiile să poată fi prezentate eficient. Tipul de grafic selectat depinde în mod esenţial de natura datelor prelucrate.
Graficele de tip bare sau arie (bar, area) sunt utile pentru vizualizarea unor rezultate, compararea lor şi afişarea unei contribuţii individuale din total.

- Graficele de tip statistic (pie charts) indică contribuţiile individuale dintr-un total.
- Histogramele (histogram) sunt utile pentru a indica distribuţia valorilor datelor.
- Graficele de tip stem şi stairstep sunt utile pentru date discrete.
- Graficele compass, feather, quiver sunt utile pentru plotarea vectorilor de tip direcţie şi viteză.
- Graficele de tip contur (contour) sunt utile la reprezentarea unor regiuni de valori egale ale datelor.
- Plotările interactive (interactive) permit selectarea unor puncte de plotare în mod interactiv.
- Graficele de tip animaţie (animations) adaugă date la grafice consecutive şi creează o animaţie.

Grafice de tip Bar şi Area

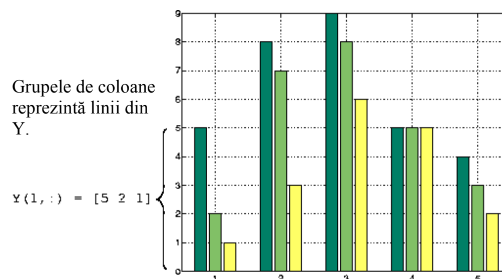
Funcţie	Descriere
Bar	Afişează coloanele unor matrici mxn ca m grupe de n bare verticale
Barh	Afişează coloanele unor matrici mxn ca m grupe de n bare orizontale
bar3	Afişează coloanele unor matrici mxn ca m grupe de n bare verticale 3-D
bar3h	Afişează coloanele unor matrici mxn ca m grupe de n bare orizontale tridimen-sionale 3-D
Area	Afişează datele din vectori ca suprafeţe

Grafice Bar grupate

Ca setare implicită, un grafic cu bare reprezintă fiecare element dintr-o matrice cu o singură bară. În cazul unui grafic 2-D, barele create cu funcţia bar sunt distribuite de-a lungul axei x, cu fiecare element dintr-o coloană desenat la altă locaţie. Toate elementele dintr-o linie sunt reprezentate grupat la aceeaşi locaţie pe axa x.

Exemplu:

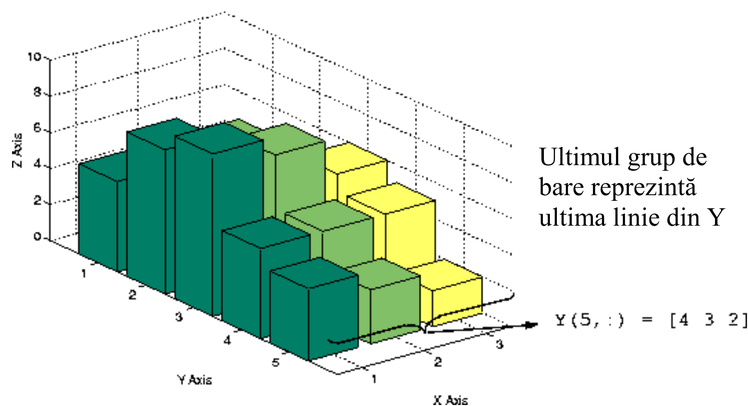
```
Y = [5 2 1
     8 7 3
     9 8 6
     5 5 5
     4 3 2];
bar(Y)
```



Grafice Bar 3-D separate

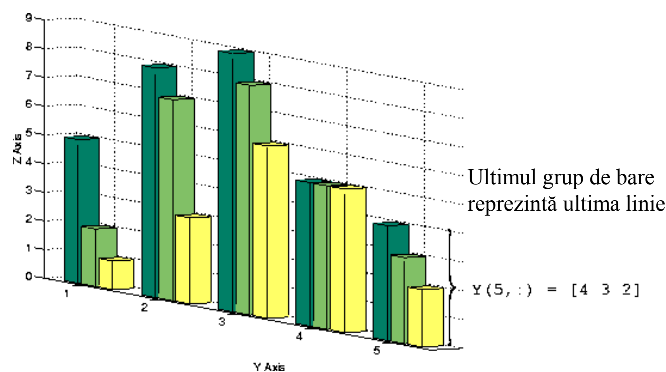
Funcţia `bar3`, în cea mai simplă formă, trasează fiecare element ca un bloc separat de tip 3-D, cu elementele fiecărei coloane distribuite de-a lungul axei y. Barele care reprezintă elementele din prima coloană a unei matrice sunt centrate la 1 pe axa x ş.a.m.d. Barele care reprezintă elementele din ultima coloană sunt centrate la valoarea size(Y,2) de pe axa x.

Exemplu: `bar3(Y).`



Grafice Bar 3-D grupate

Pentru a realiza un grafic de bare grupate 3 D se specifică argumentul '`group`':
```
bar3(Y,'group')
```



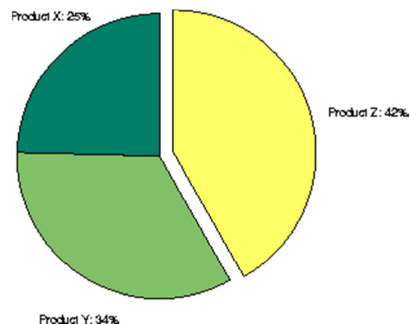
Grafice statistice - pie charts

Graficele `pie` afişează procentul cu care fiecare element al unui vector sau matrice contribuie la suma tuturor elementelor. Funcţiile `pie` şi `pie3` creează grafice 2-D şi 3-D.

În continuare prezentăm un exemplu de vizualizare a ponderii a trei produse din totalul vânzărilor. Se consideră o matrice X, pentru care fiecare coloană reprezintă vânzările anuale pentru câte un produs, pe o perioadă de înregistrări de 5 ani:
```
X = [19.3 22.1 51.6;
     34.2 70.3 82.4;
     61.4 82.9 90.8;
     50.5 54.9 59.1;
     29.4 36.3 47.0];
```
Se pot calcula vânzările pentru fiecare produs în cei 5 ani cu ajutorul funcţiei:
```
>>x = sum(X);
```
Dacă utilizăm argumentul de intrare explode putem reprezenta într-un mod explodat care dintre produse a avut o contribuţie mai mare la vânzări (de exemplu).

Programul are următoarea formă:

```
explode = zeros(size(x));
[c,offset] = max(x);
explode(offset) = 1;
h = pie(x,explode); colormap summer
```

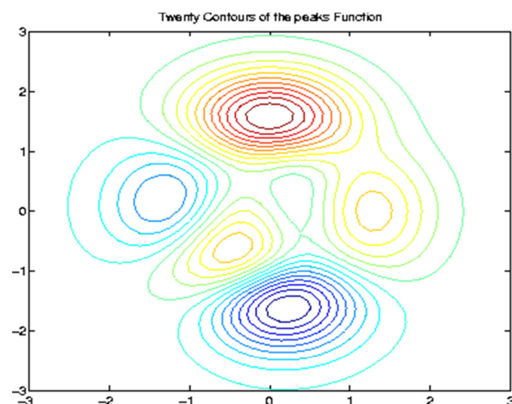


Crearea de grafice tip contur

Funcţiile contour şi contour3 afişează contururi `2-D` şi `3-D`. Funcţiile cer un singur argument, şi anume o matrice, ale cărei date sunt interpretate ca înălţimi faţă de un plan.

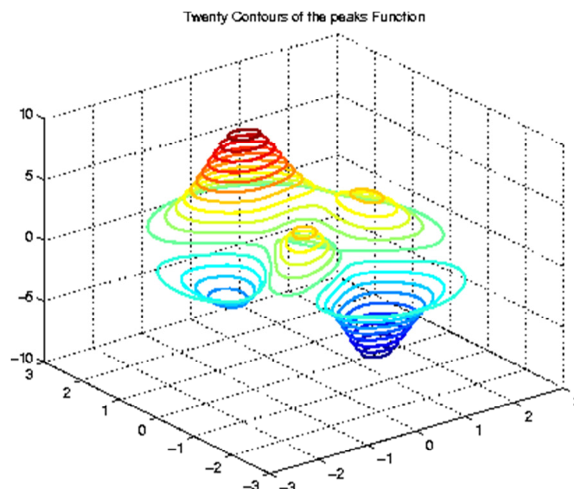
Pentru a seta numărul de niveluri de contur (implicit se realizează automat pe baza valorilor minime şi maxime) se foloseşte un argument suplimentar opţional. De exemplu,

```
[X,Y,Z] = peaks;
contour(X,Y,Z,20)%
```
afişează 20 de contururi ale funcţiei peaks într-o vedere bidimensională.



Dacă dorim o reprezentare 3 D putem folosi comenzile:

```
[X,Y,Z] = peaks;
contour3(X,Y,Z,20)
h = findobj(`Type','patch');
set(h,'LineWidth',2)
title('Twenty Contours of the peaks Function')
```



6.2. Mişcarea şi Animaţia imaginilor

Se pot crea secvenţe animate în MATLAB [7] pe două căi:

- Salvarea unui număr de imagini şi rularea lor ca pe un film.
- Ştergerea continuă şi redesenarea unor obiecte pe ecran, făcând schimbări în mod incremental la fiecare redesenare.

Pentru a realize mişcarea şi animaţia imaginilor (cadrelor) reprezentate într-o matrice M, trebuie ca într-o primă etapă să se creeze o nouă matrice cu coloanele deplasate, utilizând funcţia *getframe* care se apelează cu una din următoarele sintaxe:

```
M=getframe %returnează un vector coloană cu un cadru mişcat, care re-
prezintă o fotografiere instantanee (pixel cu pixel)
M=getframe(h)%returnează un identificator care permite modificarea
proprietăţilor obiectului figură sau a obiectului axe;
M=getframe(h,poz)%returnează un cadru mişcat al obiectului h din zona
încadrată de dreptunghiul cu dimensiunile (L,H) situate la distanţa
(l,h) de colţul din stânga jos al ferestrei grafice. Parametrul poz
trebuie să fie unn vector linie cu patru elemente: poz=[l h L H]
```

Pentru a evita utilizarea unei memorii considerabile, este preferabilă prealocarea unei matrici de mişcare capabilă să memoreze cele `n` cadre succesive, acest lucru se realizează cu funcţia:
`M=moviein(n);`
După aceste prelucrări asupra matricei ce reprezintă imaginea de animat, se utilizează funcţia **`movie`** pentru a mişca cadrele într-o anumită ordine, cu o anumită frecvenţă şi un interval de timp precizat.

Exemplu [7 , pag. 257]

Să se scrie un program care să realizeze vibraţia funcţiei predefinite `vibes` (membrană în formă de L)

Codul Matlab :

```
load vibesdat
n=max(size(L1));nh=fix(n/2);
```

```
x=(-nh:nh)/nh;
clear c;clf; view(3);
for k=1:12
    eval(['c(k)=L' num2str(k) '(24,13)/3'])
end
colormap(prism(6));
delt=0.1;ncadre=10;
M=moviein(ncadre);
for k=1:ncadre
    disp(k);
    t=k*delt;s=c.*sin(sqrt(lambda)*t);
 L=s(1)*L1+s(2)*L2+s(3)*L3+s(4)*L4+s(5)*L5+s(6)*L6+s(7)*L7+...
    s(8)*L8+s(9)*L9+s(10)*L10+s(11)*L11+s(12)*L12;
s=s.*lambda;
V=s(1)*L1+s(2)*L2+s(3)*L3+s(4)*L4+s(5)*L5+s(6)*L6+s(7)*L7+...
    s(8)*L8+s(9)*L9+s(10)*L10+s(11)*L11+s(12)*L12;
V(1:nh,1:nh)=NaN*ones(nh,nh);
cla;surfc(x,x,L,V);
axis ([-1 1 -1 1 -1 1]);
grid on
M(:,k)=getframe;
end
movie(M,20,100);
```

Concluzie

Pentru realizarea unui film se parcurg trei etape:

1. Se foloseşte `moviein` pentru iniţializarea memoriei pentru o matrice suficient de mare.
2. Se utilizează `getframe` pentru a genera fiecare cadru de film, care este returnat ca un vector coloană cu care se poate construi o matrice de tip film.
3. Se foloseşte `movie` pentru rularea "filmului" de un număr specificat de ori cu o viteză specificată.

Ştergere şi redesenare

Pot fi create diferite efecte prin selectarea unor moduri de ştergere.

Pentru crearea unei animaţii sunt utile trei moduri de ştergere:

- `none` - MATLAB® nu şterge obiectele.
- `background` - MATLAB® şterge obiectul şi îl redesenează în background. Acest mod şterge obiectul şi tot ce este sub el (linii de grid etc.).
- `xor` – Acest mod şterge doar obiectul şi este cel mai folosit la animaţie.

Pentru vizualizarea unor efecte de animaţie şi construirea unor exemple proprii este indicată utilizarea facilităţii demo a MATLAB-ului.

6.3. Reprezentări grafice tridimensionale (3 D)

Paşii tipici care trebuie parcurşi pentru trasarea unor grafice tridimensionale sunt prezentaţi în continuare.

Etapa	Instrucţiuni
1.Pregătirea datelor	Z = peaks(20);
2. Selectarea ferestrei grafice şi poziţionarea graficului în fereastră	figure(1) subplot(2,1,2)
3. Apelarea unei funcţii de plotare 3-D	h = surf(Z);
4a. Setarea unei hărţi de culori şi a unui algoritm de umbrire	colormap hot shading interp set(h,'EdgeColor','k')
4b. Adăugarea unei iluminări	light('Position',[-2,2,20]) lighting phong material([0.4,0.6,0.5,30]) set(h,'FaceColor',[0.7 0.7 0],... 'BackFaceLighting','lit')
5. Setarea unui punct de vizualizare	view([30,25]) set(gca,'CameraViewAngleMode','Manual')
6. Setarea limitelor axelor şi a marcajelor	axis([5 15 5 15 -8 8]) set(gca'ZTickLabel','Negative||Positive')
7. Setarea proporţional-ităţii	set(gca,'PlotBoxAspectRatio',... [2.5 2.5 1])

6.3.1. Reprezentarea unei matrice ca o suprafaţă

MATLAB® -ul defineşte o suprafaţă prin coordonatele z ale punctelor de deasupra unui caroiaj dreptunghiular în planul `x-y`. Graficul este format prin unirea punctelor adiacente cu linii drepte. Plotările de suprafeţe sunt utile pentru vizualizarea matricilor care sunt prea mari pentru a fi afişate în formă numerică şi pentru trasarea graficelor funcţiilor de două variabile.
MATLAB® -ul poate crea diferite forme de trasare a suprafeţelor:

Funcţie	Utilizare
mesh, surf	Trasare a unei suprafeţe
meshc, surfc	Trasarea suprafeţei, inclusiv conturul
meshz	Trasarea suprafeţei, inclusiv planul de referinţă
pcolor	Plotare plană a suprafeţei (valorile sunt proporţionale doar cu culoarea)
surfl	Trasarea suprafeţei luminată din direcţia specificată
surface	Funcţie de nivel scăzut pentru crearea unor obiecte tip grafice suprafaţă

6.3.2. Grafice realizate cu mesh şi surf

Comenzile ***mesh*** şi ***surf*** generează suprafeţe 3-D din datele provenite de la matrici. Dacă Z este o matrice pentru elementele căreia Z(i,j) se defineşte înălţimea unei suprafeţe peste un ***caroiaj (i,j),*** atunci:

```
mesh(Z)%generează o imagine colorată, caroiată a suprafeţei şi o afişează în vedere 3-D.
```

Similar,

```
surf(Z)% generează o imagine colorată, continuă a suprafeţei şi o afişează în vedere 3-D.
```

În cazul comenzii mesh se pot folosi comenzi de tipul shading pentru eliminarea liniilor de tip mesh (`shading flat`) sau pentru interpolarea umbririlor de-a lungul faţetelor suprafeţei (`shading interp`).

6.3.3. Vizualizarea funcţiilor de două variabile

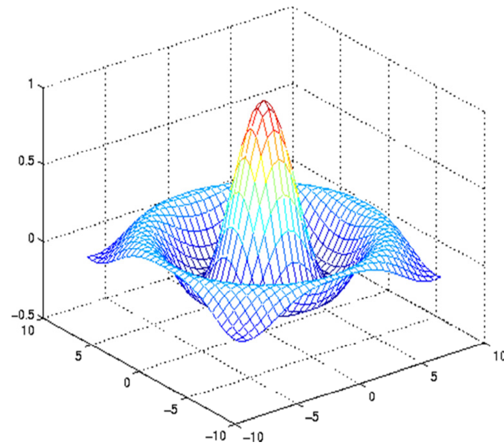
Primul pas care trebuie parcurs pentru trasarea graficului unei funcţii de două variabile,
z = f(x,y), este de a genera matricile X şi Y care definesc domeniul în care va fi vizualizată funcţia. Apoi se utilizează aceste matrici pentru evaluare şi trasarea graficului funcţiei.
`Funcţia meshgrid` transformă domeniul specificat prin doi vectori, x şi y, în matricile X şi Y. Liniile matricei X sunt copii ale vectorului x şi coloanele matricei Y sunt copii ale vectorului y.
Pentru a vedea cum se foloseşte meshgrid, vom considera `funcţia f(r)= sin(r)/r (numită funcţia sinc).`

Pentru a evalua funcţia între -8 şi 8 şi pentru x şi pentru y, este necesar doar un argument de tip vector pentru meshgrid, care va fi utilizat în ambele direcţii:

```
[X,Y] = meshgrid(-8:.5:8);
R = sqrt(X.^2 + Y.^2) + eps;
```

Matricea R conţine distanţele de la centru (originea), iar `eps` este adăugat pentru a evita împărţirea la zero. Acum se poate forma ***funcţia sinc*** şi se poate realiza plotarea cu mesh.

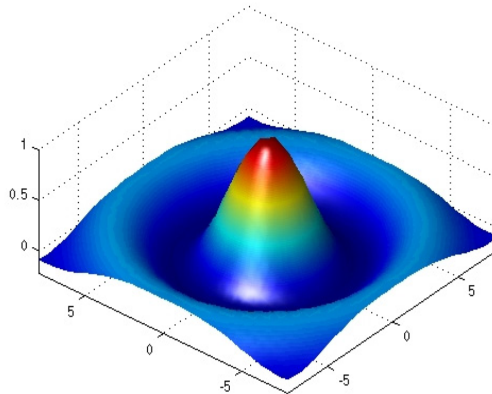
```
Z = sin(R)./R;
mesh(Z)
```



Se poate realiza o îmbunătăţire a reprezentării grafice în condiţiile utilizării aceloraşi date, prin folosirea unor facilităţi de iluminare şi ajustare a imaginii (`daspect, axis, camlight, view`).

Exemplu:

```
surf(X,Y,Z,'FaceColor','interp','EdgeColor','none',...
  'FaceLighting','phong')
daspect([5 5 1]);axis tight;view(-50,30);camlight left
```



6.3.4. Harta culorilor

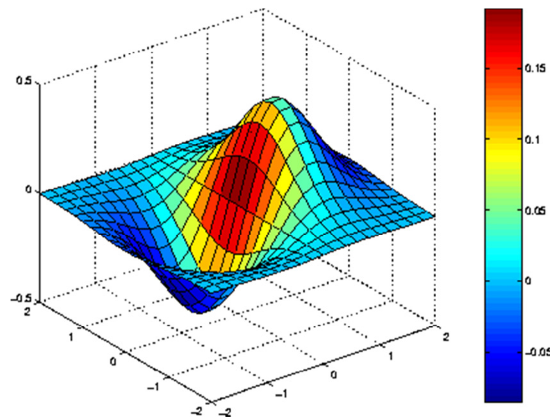
Fiecare fereastră grafică MATLAB® are asociată o hartă a culorilor (colormap), care este o matrice cu trei coloane a căror lungime este egală cu numărul de culori definite.

Fiecare linie a matricii defineşte o culoare particulară prin specificarea a trei valori în domeniul 0 – 1. Aceste valori definesc componentele RGB (red, green, blue) (adică intensităţile componentelor video roşu, verde şi albastru).

Funcţia `colormap` fără argumente returnează harta figurii curente. Funcţia `colorbar` afişează în fereastra grafică harta curentă a culorilor, sub forma unei bare aşezate lângă grafic.

Exemplu:

```
[x,y] = meshgrid([-2:.2:2]);Z = x.*exp(-x.^2-y.^2);
surf(x,y,Z,gradient(Z));colorbar
```



6.2. Handle Graphics şi Interfeţe Grafice în MATLAB (GUI)

Crearea şi manipularea graficelor în MATLAB® se realizează cu ajutorul unui sistem de grafică orientată pe obiecte denumit Handle Graphics [13]. Acest sistem furnizează componentele necesare generării unor grafice: comenzi de trasare a liniilor, textelor, grafice 3-D, poligoane etc. precum şi instrumente interactive de tipul meniurilor, butoanelor, ferestre de dialog etc.

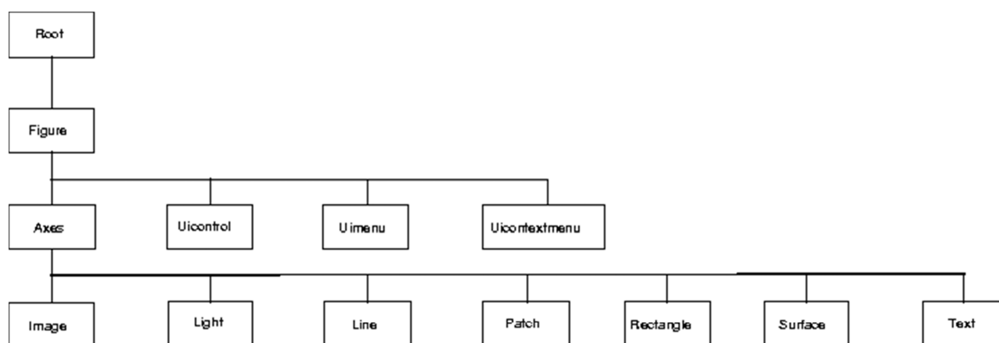
Cu Handle Graphics se pot manipula direct elementele grafice (aşa cum o fac funcţiile MATLAB® de nivel înalt descrise în capitolul anterior) pe două căi:

- fie de la linia de comandă MATLAB® ,
- fie cu ajutorul unor fişiere MATLAB® create special.

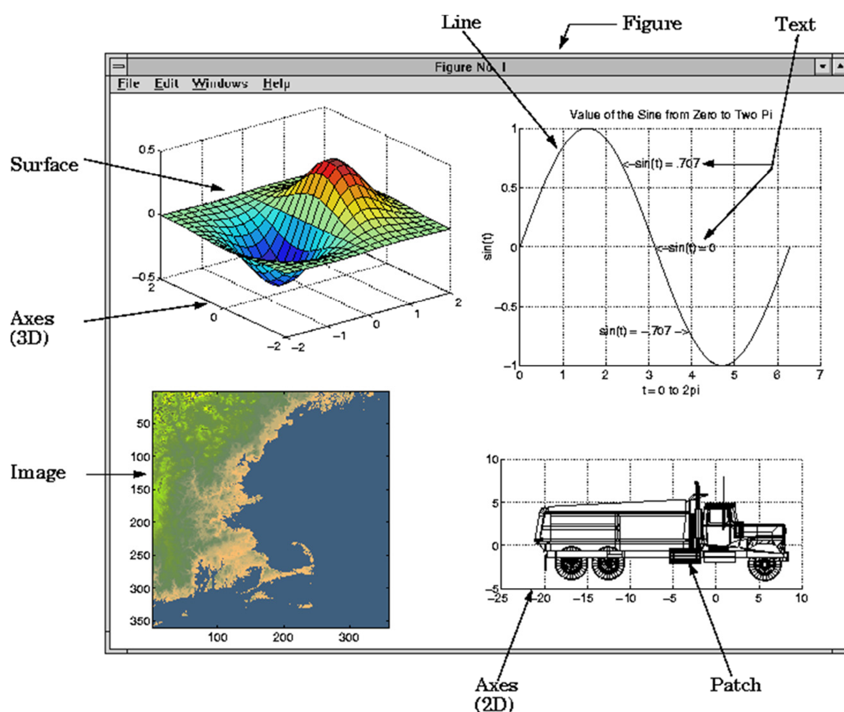
6.2.1. Ierarhia graficelor private ca obiecte

Obiectele grafice sunt de fapt elementele grafice de bază utilizate de MATLAB® pentru afişarea datelor şi pentru crearea Interfeţelor Grafice Utilizator (Graphical User Interfaces - GUI) [13]. Fiecare stare a unui obiect este asociată unui identificator unic numit handle, care poate fi folosit pentru manipularea caracteristicilor obiectului respectiv (caracteristici care sunt numite proprietăţile obiectului).

Obiectele grafice sunt structurate într-o ierarhie pe trei nivele:



Ierarhia este bazată pe interdependenţele dintre diferitele obiecte grafice. De exemplu, pentru trasarea unui obiect linie, MATLAB® utilizează un obiect de tip axe pentru orientarea şi furnizarea unui sistem de referinţă liniei. Obiectul de tip axe are nevoie la rândul său de o fereastră grafică pentru afişarea liniei.
Obiectele grafice sunt interdependente şi prin urmare un ecran grafic conţine o mare varietate de obiecte care împreună furnizează o imagine sau un grafic care are o semnificaţie clară. Pentru exemplificare se poate analiza următoarea fereastră grafică, fereastră care conţine mai multe obiecte grafice.



Fiecare tip de obiect grafic are o funcţie generatoare corespunzătoare, funcţie care este utilizată pentru crearea unui obiect din clasa respectivă de obiecte. Funcţiile de generare a obiectelor au aceleaşi nume ca şi obiectele pe care le creează (funcţia text pentru obiecte de tip text, funcţia figure pentru obiecte de tip figură etc.).

Tipurile de obiecte grafice sunt descrise pe scurt în continuare [19].

6.2.2. Rădăcina (Root)

În fruntea ierarhiei este obiectul rădăcină, care corespunde cu ecranul calculatorului. Acest obiect nu trebuie creat, el există, este unic şi toate celelalte obiecte sunt descendenţii acestuia. Se pot modifica anumite proprietăţi ale obiectului rădăcină.

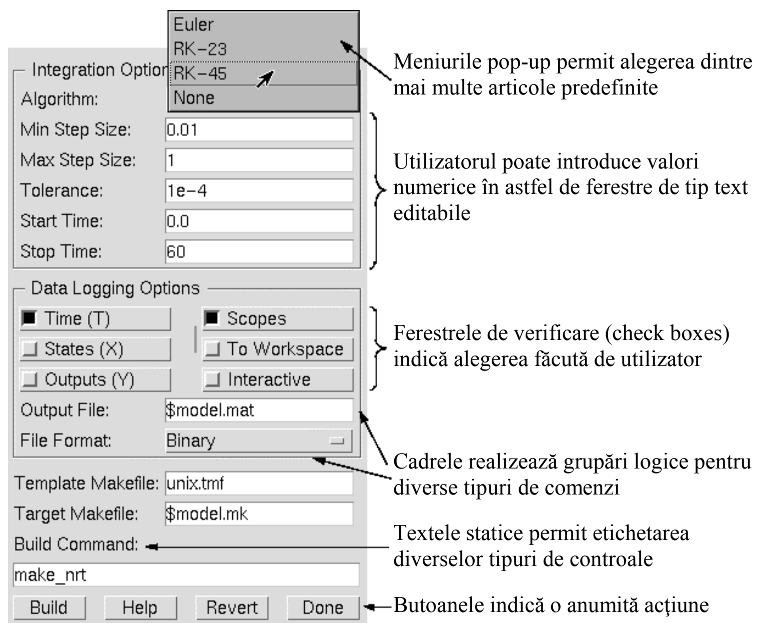
Obiectele figură (Figure)

Obiectele de tip figură sunt ferestre individuale pe ecranul rădăcină pe care MATLAB® -ul afişează graficele. Nu există limită pentru numărul de ferestre grafice (decât cele datorate limitelor calculatorului). Toate obiectele figură sunt "copii" ai rădăcinii şi celelalte obiecte grafice sunt descendenţi ai figurilor.

Obiectele de tip Uicontrol

Obiectele Uicontrol sunt elemente de control ale interfeţei utilizator care execută subrutine de apel atunci când utilizatorul activează un obiect. Există mai multe stiluri de control cum ar fi butoane, liste etc. Fiecare astfel de instrument este proiectat să accepte un anumit tip de informaţie de la utilizator. De exemplu, listele sunt de obicei folosite pentru pentru furnizarea unei liste de nume, din care utilizatorul poate selecta unul sau mai multe articole.
Obiectele Uicontrol pot fi utilizate în diferite combinaţii pentru construirea unor ecrane de control şi a unor ferestre de dialog. În exemplul următor sunt prezentate astfel de combinaţii:

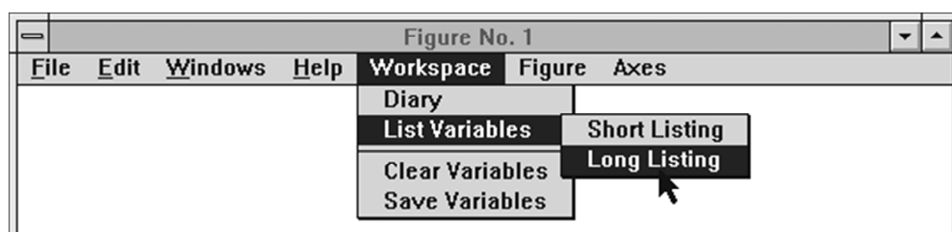
- meniuri "pop-up",
- ferestre de tip text editabile,
- ferestre de verificare (check boxes),
- butoane,
- text static,
- cadre etc.



Obiectele Uicontrol sunt copiii obiectelor de tip figură şi sunt independente de obiectele de tip axe.

Obiectele de tip Uimenu

Obiectele Uimenu sunt meniuri “pull-down” care execută rutine de apelare atunci când utilizatorul selectează un articol individual dintr-un meniu. MATLAB-ul plasează obiectele Uimenu pe bara de meniuri a ferestrei grafice, la dreapta meniurilor existente definite de sistem.
Imaginea următoare [14] arată partea de sus a unei figuri MS-Windows care are definite trei meniuri de top Uimenu (intitulate Workspace, Figure, şi Axes). Două niveluri de submeniuri sunt vizibile în meniul Workspace.



Obiectele Uimenu sunt descendente directe ale obiectelor de tip figură şi deci sunt independente de axe.

Obiectele de tip axe (Axes)

Obiectele de tip axe definesc o regiune într-o fereastră de tip figură şi orientează descendenţii lor spre această regiune. Obiectele de tip axe sunt copiii obiectelor de tip figură şi sunt părinţii obiectelor de tip imagine, luminozitate, linie, patch, suprafaţă şi text.
Toate funcţiile care trasează grafice (plot, surf, mesh, bar etc.) creează un obiect de tip axe dacă nu există deja unul.
Uicontrol şi Uimenu nu sunt descendenţi ai obiectelor de tip axe.

Obiectele de tip Imagine (Image)

O imagine în MATLAB constă într-o matrice de date şi o hartă a culorilor. Există trei tipuri de bază de imagini care diferă în funcţie de modul în care elementele matricii de date sunt interpretate ca pixeli color – indexaţi, intensitate şi “truecolor”.

Obiectele de tip lumină (luminozitate) (Light)

Aceste obiecte definesc surse de lumină care influenţează toate obiectele de tip patch şi suprafaţă dintre axe. Se pot seta proprietăţile care determină tipul sursei de lumină, culoarea, localizarea şi altele.

Obiectele de tip linie (Line)

Obiectele de tip linie sunt elemente grafice de bază care sunt folosite pentru a genera cele mai multe plotări 2-D şi unele 3-D. Funcţiile de nivel înalt plot, plot3, loglog etc. generează obiecte de tip linie. Sistemul de coordonate al obiectului părinte – obiectul de tip axe – poziţionează şi orientează linia.

Obiectele de tip Patch

Aceste obiecte sunt contururi poligonale cu muchii (laturi), umplute. Un singur obiect patch poate conţine mai multe feţe, fiecare colorată independent. Funcţiile fill, fill3, contour3 creează obiecte patch. Ca şi în cazul liniei, sistemul de coordonate al obiectului părinte (axele) poziţionează şi orientează obiectul patch.

Obiectele de tip dreptunghi (Rectangle)

Aceste obiecte sunt arii 2-D umplute, cu o formă care poate varia de la un dreptunghi la o elipsă.

Obiectele de tip suprafaţă (Surface)

Obiectele de tip Surface sunt reprezentări 3-D ale matricilor de date create prin plotarea fiecărui element al matricii ca o înălţime deasupra planului x-y. MATLAB-ul poate trasa suprafeţe pline, colorate sau doar o reţea de linii (mesh) care conectează punctele respective. Sistemul de coordonate al axelor poziţionează şi orientează obiectul de tip suprafaţă. Funcţiile de nivel înalt pcolor, surf, mesh generează obiecte de tip suprafaţă.

Obiectele de tip Text

Obiectele de tip text sunt de fapt şiruri de caractere. Sistemul de coordonate al axelor poziţionează textul. Funcţiile de nivel înalt: `title`, `xlabel`, `ylabel`, `zlabel`, `gtext` generează obiecte de tip text.

6.2.2. Proprietăţile obiectelor grafice

Proprietăţile obiectelor grafice determină aspectul şi comportamentul acestora. Proprietăţile includ informaţii generale (tipul obiectului, părinte, copii, dacă obiectul este vizibil etc.) şi informaţii specifice unei anumite clase particulare de obiecte.
MATLAB® -ul organizează informaţiile într-o ierarhie şi salvează aceste informaţii în proprietăţi ale obiectelor. De exemplu, proprietăţile rădăcinii conţin identificatorul (handle) figurii curente şi locaţia curentă a pointerului (cursorului), proprietăţile figurii conţin liste cu descendenţii şi evenimentele din fereastră, proprietăţile axelor conţin informaţii despre cum fiecare din obiectele copil foloseşte harta culorilor etc.
Valoarea curentă a oricărei proprietăţi poate fi aflată, iar unele valori pot fi modificate. Valoarea unei proprietăţi este aplicată numai unui obiect particular şi nu întregii clase de obiecte. Se pot seta valori implicite care să fie valabile pentru toate obiectele create ulterior.
Anumite proprietăţi sunt comune tuturor obiectelor grafice.

Proprietate	Informaţii conţinute
BusyAction	Controlează modul în care MATLAB-ul apelează rutinele de întreruperi definite pentru un anumit obiect.
ButtonDownFcn	Rutină executată la apăsarea unui buton.
Children	Manipulează toate obiectele copil ale obiectului.
Clipping	Activare/dezactivare mod tăiere.
CreateFcn	Rutină executată atunci când acest tip de obiect este creat.

DeleteFcn	Rutină executată atunci când se dă o comandă de distrugere (ştergere) a obiectului.
HandleVisibility	Permite controlul obiectului de la linia de comandă sau din rutine de apelare.
Interruptible	Determină când o rutină poate fi întreruptă printr-o rutină invocată ulterior.
Parent	Părintele obiectului.
Selected	Indică dacă obiectul este selectat.
SelectionHighlight	Specifică dacă este indicată starea de selectare.
Tag	Etichetă a unui obiect specificată de utilizator.
Type	Tipul obiectului (figură, linie, text etc.)
UserData	Orice dată care se doreşte a fi asociată obiectului.
Visible	Determină dacă obiectul este vizibil sau nu.

6.2.3. Funcţii de generare a obiectelor grafice

Fiecare obiect grafic, mai puţin rădăcina, are o funcţie de generare corespondentă:

Funcţie	Descriere obiect
axes	Sistem de coordonate carteziene care scalează şi orientează obiectele copil: imagine, lumină, linie, patch, suprafaţă şi text.
figure	Fereastră pentru afişare grafică.
image	Imagine 2-D definită prin indicarea hărţii culorilor sau valori RGB. Datele pot fi pe 8 biţi sau dublă precizie.
light	Sursă direcţionată de lumină, localizată între axe, care influenţează suprafeţele şi obiectele patch.
line	Linie formată prin conectarea coordonatelor prin segmente drepte într-o secvenţă specificată.
patch	Formă poligonală creată prin interpretarea fiecărei coloane din matricile de coordonate ca un poligon separat.
rectangle	Arie 2-D umplută (plină), cu formă de la dreptunghi la elipsă.
surface	Suprafaţă cu feţe dreptunghiulare, definite prin interpretarea elementelor matricei ca înălţimi deasupra planului.
text	Şir de caractere localizat în sistemul de coordonate al axelor.
uicontextmenu	Meniu context ce poate fi asociat cu alt obiect grafic.
uicontrol	Interfaţă utilizator programabilă (butoane, liste etc.).
uimenu	Meniu programabil care apare în partea superioară a figurii.

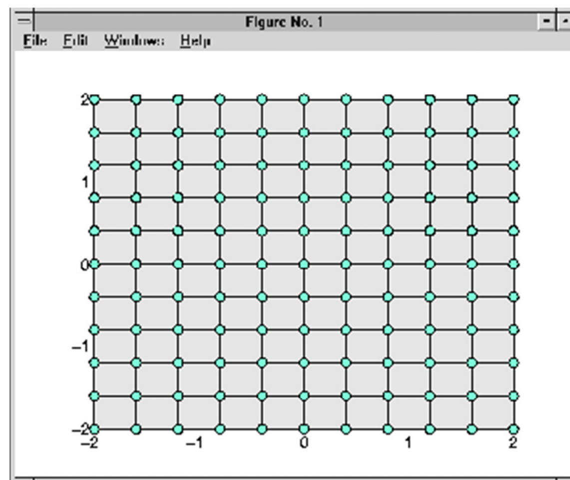
Toate funcţiile de generare a obiectelor au un format similar:

```
handle=function('propertyname',propertyvalue,...)
```

Exemplu de generare a obiectelor grafice

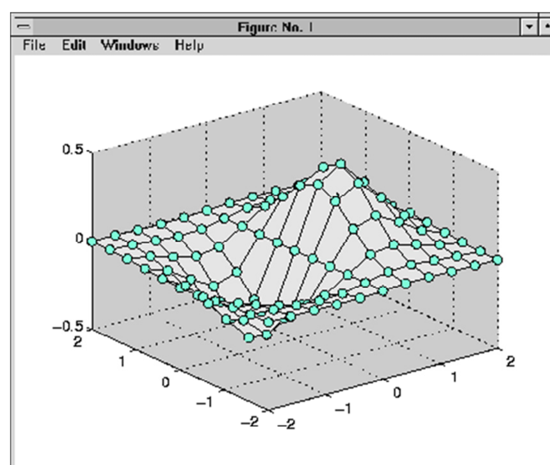
În exemplul următor este evaluată o funcţie matematică şi sunt create trei obiecte grafice folosind valorile proprietăţilor specificate ca argumente ale comenzilor `figure`, `axes` şi `surface`, celelalte proprietăţi având valori implicite.

```
 [x,y] = meshgrid([-2:.4:2]);
Z = x.*exp(-x.^2-y.^2);
fh=figure('Position',[350 275 400 300],'Color','w');
ah=axes('Color',[.8 .8 .8],'XTick',[-2 -1 0 1 2],...
          'YTick',[-2 -1 0 1 2]);
sh = surface('XData',x,'YData',y,'ZData',Z,...
             'FaceColor',get(ah,'Color')+.1,...
             'EdgeColor','k','Marker','o',...
             'MarkerFaceColor',[.5 1 .85]);
```



Funcţia `surface` nu foloseşte o vedere 3-D ca funcţia de nivel înalt surf. Se poate schimba vederea într-una 3-D cu camera commands sau cu comanda view:

```
view(3)
```



6.2.4. Construcţia interactivă a unei interfeţe grafice cu utilizatorul folosind GUI

Pentru a construi o interfaţă grafică cu utilizatorul de tip GUI se poate face apel la facilităţile MATLAB® dezvoltate în acest sens şi prezentate în **`Matlab Guide (Graphical User Interfaces Development Environment).`**
Tastăm în linia de comandă
```
>> guide
```
În cutia de dialog se selectează ***Blank GUI (Default)*** în câmpul ***GUIDE Templates.*** Se activează opţiunea *Save on startup as,* iar în fereastra alăturată se introduce **roger.fig** şi apoi tastăm butonul ***OK*** al căsuţei de dialog.

În urma acestei acţiuni va apare o fereastră ce reprezintă mediul de dezvoltare ***Matlab Guide.*** În zona din stânga ecranului sunt dispuse obiectele predefinite disponibile în Matlab . Fiecare stare a unui obiect este asociată unui identificator unic numit *handles,* care poate fi folosit pentru manipularea caracteristicilor obiectului respective (caracteristici care sunt numite ***proprietăţile obiectului).***
Etapa 1
Salvarea interfeţei grafice **roger.fig** se face printr-un clic pe meniul ***File+Save As***. În urma acestei operaţiuni de salvare Matlab crează două fişiere **roger.fig** şi **roger.m** cel de-al doilea fiind automat deschis.
Acest ultim fişier conţine codul specific interfeţei grafice unde trebuiesc adăugate instrucţiunile de execuţie a anumitor comenzi de interacţiune a utilizatorului cu programul.
Pentru a schimba proprietăţile obiectele se va executa dublu click pe rând pe fiecare obiect pentru a deschide fereastra ***Property Inspector*** specific fiecărui obiect grafic.
Etapa 2
Scrierea liniilor de program pentru obiectele GUI (Callbacks), în continuare salvează modificările efectuate în fişierul M şi de pe linia de comandă a ferestrei ***Command Window*** se tastează
```
>> roger
```
care va lansa în execuţie interfaţa grafică astfel creată.
Observaţie: Codul MATLAB® se poate modifica după preferinţele utilizatorului.

CAPITOLUL 7.

ALGEBRĂ

7.1. Matricile în MATLAB®

Definiţe: În Matlab, o matrice este un tablou dreptunghiular de numere. Scalarii de exemplu sunt matrici 1 x 1, iar matricile cu o singură linie sau coloană sunt de fapt vectori.

7.1.1. Introducerea matricilor

Matricile se pot introduce în mai multe moduri:

- Introducerea unei liste explicite cu elementele matricei.
- Încărcarea unor date din fişere externe de date.
- Generarea de matrici utilizând funcţii built-in.
- Crearea de matrici în fişierele M-files.

Vom da exemple folosind matricea lui Dürer cunoscută ca şi matricea magică, deoarece suma elementelor pe linii şi coloane este egală cu suma elementelor de pe diagonala principală şi secundară.
Trebuie respectate câteva convenţii simple:

- Elementele unei linii sunt separate prin virgule sau spaţii. Sfârşitul unei linii se marchează cu punct şi virgulă.
- Lista de elemente care formează matricea se delimitează cu paranteze drepte [7]:

Pentru introducerea matricii lui Dürer tastăm:

```
» A = [16 3 2 13; 5 10 11 8; 9 6 7 12; 4 15 14 1]
```

MATLAB® -ul va afişa matricea:

```
A =
    16     3     2    13
     5    10    11     8
     9     6     7    12
     4    15    14     1
```

O dată introdusă, matricea este memorată în workspace (spaţiul de lucru) şi poate fi apelată simplu, ca A.Verificăm proprietăţile fundamentale ale matricei A folosind comenzile MATLAB®:
sum, transpose, diag
Suma elementelor de pe cele 4 coloane se calculează rapid cu:

```
» sum(A)
ans =
    34    34    34    34
```

Pentru calcularea sumelor pe linii, efectuăm întâi transpunerea matricii şi apoi aplică din nou comanda **`sum.`**

Transpusa se calculează cu:
```
» A'
ans =
    16     5     9     4
     3    10     6    15
     2    11     7    14
    13     8    12     1
```
şi apoi
```
» sum(A')'
ans =
    34
    34
    34
    34
```
Suma elementelor de pe diagonală se calculează cu tot cu funcţia sum, dar după ce în prealabil vom sorta cu funcţia diag elementele de pe diagonala principală:
```
» diag(A)
ans =
    16
    10
     7
     1

» sum(diag(A))
ans =
   34
```
Un anume element al matricii, de exemplu elementul din linia i coloana j se notează A(i,j). Prin urmare o altă cale (mai puţin rapidă) de a calcula suma de pe patra coloană de exemplu este următoarea:
```
» A(1,4) + A(2,4) + A(3,4) + A(4,4)
ans =
      34
```
Dacă specificăm un element care nu există în matrice, primim un mesaj de eroare:
```
» t = A(4,5)
Index exceeds matrix dimensions.
```

7.1.2. Operatorul :

Operatorul ':' este foarte important. De exemplu, expresia
```
» 1:10
```
este un vector linie
```
ans =
            1    2    3    4    5    6    7    8    9   10
```
Alte exemple:
```
» 100:-7:50
ans =
            100    93    86    79    72    65    58    51
» 0:pi/4:pi
ans =
            0    0.7854    1.5708    2.3562    3.1416
```

7.1.3. Expresia :

A (1:k, j) se referă la primele k elemente ale coloanei j a lui A.
Dacă este utilizat în paranteze operatorul : atunci înseamnă că ne referim la toate elementele unei linii sau coloane

```
» sum(A(:,3))
calculează suma elementelor din coloana a treia a lui A:
ans =
      34
```

O altă proprietate interesantă a pătratului magic este că suma magică 34 este obţinută şi prin însumarea elementelor matricii şi prin împărţirea la dimensiunea matricii (4):

```
» sum(1:16)/4
ans =
      34
```

Observaţie:
Suma magică pentru orice pătrat magic n x n este (n3 + n)/2 (se poate calcula şi cu ajutorul Symbolic Math Toolbox) [17]. MATLAB-ul operează cu matricile cu aceeaşi uşurinţă cu care lucrează cu scalarii. Pentru adunarea a două matrici de exemplu se foloseşte pur şi simplu semnul + ca la o adunare obişnuită. Bineînţeles că matricile trebuie să aibă aceleaşi dimensiuni pentru a putea fi adunate.

Exemplu:

```
» A=[2 3;15 -3]
A =
      2      3
     15     -3

» B=[11 -21; 12 4]
B =
     11    -21
     12      4

» C=A+B
C =
     13    -18
1
```

Pentru înmulţirea a două matrici se foloseşte operatorul * , valabil de altfel şi pentru operaţiile cu scalari.

Exemplu:

```
» D=A*B
D =
     58    -30
-327
```

Dacă dimensiunile matricilor care se înmulţesc nu sunt corespunzătoare, atunci va fi furnizat un mesaj de eroare:

```
» E=[1 23; -12 2;1 2]
```

```
E =
     1    23
   -12     2
     1     2

» F=A*E
??? Error using ==> *
```
Inner matrix dimensions must agree.

Pentru “depanarea” programului în cazul unor astfel de greşeli se poate utiliza comanda size care ne dă informaţii despre dimensiunile matricilor respective şi permite corectarea erorilor:

```
» size(A)
ans =
     2     2
» size(E)
ans =
2
```
O facilitate interesantă a MATLAB-ului este aceea că lucrează cu matricile cu operatori logici şi relaţionali într-un mod asemănător acestor operaţii efectuate cu scalari.

De exemplu, pentru operaţiunea scalară. Dacă dorim de exemplu să comparăm fiecare element al matricii A cu elementul corespunzător din matricea B, procedăm asemănător:

```
» L=A<=B
L =
      1      0
      0      1
```

Operatorii logici, adică & pentru ŞI (AND), | pentru SAU (OR), ~ pentru NU (NOT), vor returna valoarea 1 pentru ADEVĂRAT şi 0 pentru FALS.

Exemplu:

```
» A&B
ans =
     1     1
     1     1
» ~A
ans =
            0       0
            0       0
```

MATLAB® -ul lucrează direct cu multe operaţii matriceale: aritmetica matricilor, ecuaţii liniare, valori proprii, factorizări etc **, ele sunt localizate în directorul matfun.**

În continuare sunt prezentate câteva din funcţiile care lucrează cu matrici.

Categoria	Funcţia	Descriere
Analiza matri- ceală	norm	Norma unei matrice sau a unui vector.
	normest	Estimează norma-2 a matricei.
	rank	Rangul matricei.
	det	Determinant.
	trace	Suma elementelor de pe diagonală.
	null	Spaţiul Nul.
	orth	Ortogonalizare.
	subspace	Unghiul dintre 2 subspaţii.
Ecuaţii liniare	\ şi /	Utilizate la calculul soluţiilor ecuaţiilor liniare.
	inv	Inversa matricei.
	cond	Numărul de condiţie pentru inversare.
	chol	Factorizarea Cholesky.
	lu	Factorizarea LU.
	qr	Decompoziţia ortogonal-triunghiulară.
	pinv	Pseudoinversa.
	lscov	Cele mai mici pătrate cu covarianţă cunoscută.
Valori proprii şi şi vectori proprii	eig	Valori proprii şi vectori proprii.
	svd	Decompoziţia în valori singulare.
	poly	Polinomul caracteristic.
	hess	Forma Hessenberg.
	qz	Factorizarea QZ.
	schur	Decompoziţia Schur.
Funcţii de matrice	expm	Exponenţiala de matrice.
	logm	Logaritmul de matrice.
	sqrtm	Rădăcina pătrată de matrice.
	funm	Evaluarea unei funcţii generale de matrice.

Operaţiile elementare cu matrici au fost deja prezentate (adunarea matricilor, înmulţirea acestora, transpusa unei matrice, funcţiile sum, diag etc.).
Teorema (29, pag 120)
Fie $A \in \Re^{m \times n}$ cu $m \geq n$. Atunci există o singură matrice $m \times n$ Q orthogonală şi o unică matrice $n \times n$ R triunghiulară superior, cu elementele de pe diagonala principală pozitive astfel ca: A=QR.
Vom da un exemplu descompunere QR (Cholesky Decomposition)

Codul Matlab (29, pag121)
```
function Cholesky(A)
%apel R=Cholesky (A)
%R= matrice triunghiular superioara
[m,n]=size(A);
for k=1:m
    if A(k,k)<=0
        error('matrix is not HPD')
    end
    for j=k+1:m
        A(j,j:m)=A(j,j:m)-A(k,j:m)*A(k,j)/A(k,k);
    end
    A(k,k:m)=A(k,k:m)/sqrt(A(k,k));
end
R=triu(A);
```

În continuare sunt prezentate câteva ***matrici speciale*** utile în toate tipurile de reprezentări matematice:

7.1.4. Matrici speciale

Matricea identitate (unitate)

Este o matrice cu elementele de pe diagonala principală egale cu 1 iar restul elementelor sunt nule. Notaţia matematică I provine de la denumirea matricii şi nu este folosită în MATLAB®, pentru evitarea unor confuzii.
Se utilizează sintaxa:
`eye(m,n)`
Această funcţie returnează o matrice identitate mxn. Dacă se foloseşte:
`eye(n)`
atunci este vorba de o matrice identitate pătratică nxn.

Matricea ones

Este o matrice care are toate elementele egale cu 1. Forme posibile:
`ones(n)` este o matrice nxn cu elemente de 1
`ones(m,n)` sau `ones([m,n])` sunt matrici mxn cu elemente de 1.
`ones(size(A))` are aceeaşi dimensiune cu o matrice A şi are elemente de 1

Matricea zeros

Este o matrice care are toate elementele egale cu 0.
Forme posibile:

`zeros(n)` este o matrice nxn de zerouri
`zeros(m,n)` sau zeros([m,n]) sunt matrici mxn de 0
`zeros(size(A))` are aceeaşi dimensiune cu o matrice A şi are elemente de 0.

7.2. Rezolvarea ecuaţiilor liniare

Una din cele mai importante probleme ale calculului din domeniul tehnic este soluţionarea sistemelor de ecuaţii liniare.
Definirea problemei este pe scurt următoarea [24] :
Dacă se dau două matrici A şi B, există o matrice unică X astfel încât
AX = B sau XA = B ?
MATLAB® utilizează notaţia din cazul scalar şi pentru descrierea soluţiei unui sistem de ecuaţii liniare. Cele două simboluri utilizate în cazul scalar al diviziunii (împărţirii) şi anume slash, /, şi backslash, \, sunt folosite pentru definirea soluţiei:
X = A\B este soluţia ecuaţiei matriceale AX = B.
X = B/A este soluţia ecuaţiei matriceale XA = B.
În practică, ecuaţiile liniare de forma AX = B sunt mai des întâlnite.
Deoarece matricea A, care conţine de fapt coeficienţii sistemului, poate să nu fie pătratică ci de tipul general mxn, există trei cazuri posibile:

m = n.	Sistem pătratic. Se poate căuta o soluţie exactă.
m > n.	Sistem supradeterminat (incompatibil). Se caută o soluţie de tip cele mai mici pătrate.
m < n.	Sistem nedeterminat. Se poate căuta o soluţie cu cel mult m componente nenule.

În multe cazuri MATLAB-ul dă un diagnostic (o soluţie) automat prin examinarea coeficienţilor matricelor.
Câteva din aceste cazuri:

- Permutarea matricilor triunghiulare
- Matrici simetrice, pozitiv definite
- Matrici pătratice nesingulare
- Sisteme rectangulare supradeterminate
- Sisteme rectangulare nedeterminate

7.2.1. Sisteme pătratice

Cel mai simplu caz este cel corespunzător unei matrice pătratice A şi a unui vector coloană b. Soluţia x = A\b are aceeaşi dimensiune ca b.
Dacă A şi B sunt pătratice de aceleaşi dimensiuni atunci soluţia X = A\B are aceeaşi dimensiune ca A sau B.
Observaţie: Dacă matricea A este singulară (determinant nul) atunci soluţia ecuaţiei AX = B nu există sau nu este unică.

7.2.2. Sisteme supradeterminate (incompatibile)

Aceste tipuri de sisteme sunt des întâlnite în diverse situaţii, cum ar fi de exemplu aproximarea unor curbe din date experimentale.

7.2.3. Sisteme nedeterminate

Sistemele liniare nedeterminate au mai multe necunoscute decât ecuaţii. Dacă există şi constrângeri (restricţii) suplimentare, atunci este vorba de o problemă de programare liniară.
Operatorul backslash din MATLAB® permite căutarea soluţiei în cazul fără restricţii. Soluţia nu este niciodată unicăMATLAB® -ul găseşte o soluţie de bază (care are cel mult m componente nenule).
Găsirea soluţiei particulare se bazează pe factorizarea QR (decompoziţia ortogonal-triunghiulară).
Vom prezenta un exemplu (care utilizează funcţia matriceală `random – rand`).

```
» R = fix(10*rand(2,4))
  R =
       6     8     7     3
       3     5     4     1
  » b = fix(10*rand(2,1))
  b =
       1
       2
```

Sistemul liniar Rx = b implică două ecuaţii cu 4 necunoscute. Soluţia se poate afişa în format raţional (coeficienţii sunt numere întregi).
Soluţia particulară se obţine astfel:

```
» format rat
» p = R\b
  p =
      0
     5/7
      0
   -11/7
```

Soluţia completă a sistemului nedeterminat se obţine prin adăugarea unui vector arbitrar din spaţiul nul folosind funcţia null:

```
 » Z = null(R,'r')
   Z =
      -1/2          -7/6
      -1/2           1/2
        1             0
        0             1
```

Orice vector de forma : `x=p+Z*q` pentru `q` vector arbitrar satisface `R*x=b`.

7.3. Inverse şi determinanţi

Dacă matricea A este pătratică şi nesingulară, ecuaţiile AX = I şi XA = I au aceeaşi soluţie X. Această soluţie este chiar inversa lui A, notată matematic prin A-1, şi poate fi calculată cu funcţia `inv`.
Determinantul unei matrice se poate calcula cu funcţia `det` (trebuie acordată atenţie problemelor de scalare şi rotunjire care apar în calcule).

Exemple:
```
» A=[1 1 1;1 2 3;1 3 6];
» d = det(A)
d =
       1

» X = inv(A)
X =
       3    -3     1
      -3     5    -2
       1    -2     1
```

7.3.1. Pseudoinverse

Matricile dreptunghiulare (rectangulare) nu au inverse sau determinanţi. Atunci cel puţin una din ecuaţiile AX = I sau XA = I nu are soluţie. Se poate utiliza în acest caz o pseudoinversă care poate fi calculată cu funcţia pinv:

```
» A1=[A;[1 3 5]]

A1 =
     1     1     1
     1     2     3
     1     3     6
     1     3     5

» X=pinv(A1)

X =
    1.5000   -0.0000    1.0000   -1.5000
   -0.8333    0.6667   -2.0000    2.1667
    0.1667   -0.3333    1.0000   -0.8333
```

7.4. Funcţii de matrice. Valori proprii

7.4.1. Puteri de matrice

Dacă A este o matrice pătratică şi p este un număr întreg pozitiv, atunci `A^p` multiplică pe `A` cu ea însăşi de p ori.

```
» X = A^2
X =
     3     6    10
     6    14    25
    10    25    46
```

Dacă `A` este pătratică şi nesingulară, atunci `A^(-p)` multiplică pe `inv(A)` cu ea însăşi de p ori.

```
» Y=A^(-2)
Y =
   19.0000  -26.0000   10.0000
  -26.0000   38.0000  -15.0000
   10.0000  -15.0000    6.0000
```

Ridicarea la putere element cu element se face utilizând operatorul (funcţia) .^. De exemplu:

```
» X = A.^2
A =
     1     1     1
     1     4     9
     1     9    36
```

7.4.2. Rădăcina pătrată de matrice

Funcţia `sqrtm(A)` permite calculul lui `A^(1/2)` printr-un algoritm mai precis decât utilizarea puterii de matrice.

7.4.3. Exponenţiala de matrice

Un sistem de ecuaţii diferenţiale ordinare cu coeficienţi constanţi poate fi scris:

$$\frac{dy}{dx} = A\,x(t),$$

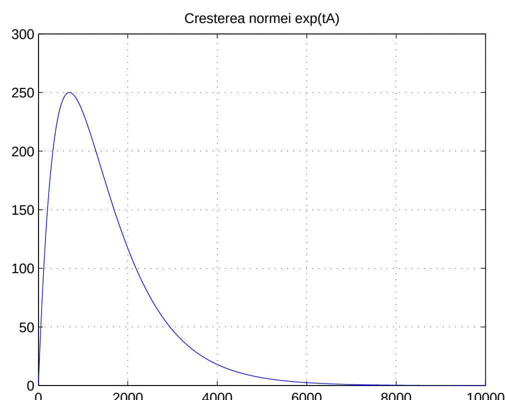
unde `x = x(t)` este un vector de funcţii de timp şi A este o matrice independentă de timp.
Soluţia sistemului poate fi scrisă prin intermediul exponenţialei de matrice
$x(t)=e^{tA}\ x(0)$.
Funcţia `expm(A)` permite calculul exponenţialei de matrice.

Exemplu.
Următorul program calculează: `norm(exp(tA))` pentru valori mari ale timpului. Datorită ne-normalităţii matricei A are loc o creştere importantă a acestei mărimi pentru valori mici ale lui t.

Codul Matalab este:

```
eps=0.001;
A=[-eps,0.;1.,-2*eps];
n=100001;
% Programul calculeaza numarul lui Henrici pentru A
HA=sqrt(norm(A'*A-A*A'));
% Calculeaza si traseaza exp(t*A) pentru matricea A ne-normala
t=0.:.1:10000;
for i=1:n
    c(i)=norm(expm(t(i)*A));
end
plot(t,c);
grid on
title('Cresterea normei exp(tA)')
```

În urma execuţiei acestui program obţinem următorul rezultat:



7.4.4. Valori proprii

O valoare proprie şi un vector propriu ale unei matrice pătratice A sunt un scalar λ şi un vector v care satisfac egalitatea :

$$Av=lv$$

Cu valorile proprii pe diagonala unei matrice de tip diagonal G şi cu vectorii proprii corespunzători care formează coloanele unei matrice V vom avea :

$$A\ V=V\ G$$

Dacă V este nesingulară obţinem descompunerea pe baza valorilor proprii:

$$A=V\ G\ V^{-1}$$

Exemplu:

```
» A=[-1 -3 1;2 -2 -1;0 1 -3]
A =
    -1    -3     1
     2    -2    -1
     0     1    -3
» lambda=eig(A)
lambda =
  -1.7593 + 2.4847i
  -1.7593 - 2.4847i
  -2.4814
```

Lambda va fi un vector care conţine valorile proprii ale matricei.
Dacă funcţia eig este utilizată cu două argumente de ieşire vom obţine vectorii proprii şi valorile proprii (acestea sub forma diagonală):

```
» [V,D]=eig(A)
V =
   0.2233 + 0.6835i   0.2233 - 0.6835i   0.3160
   0.6481 - 0.0862i   0.6481 + 0.0862i   0.4368
   0.0765 - 0.2227i   0.0765 + 0.2227i   0.8422
```

```
D =
  -1.7593 + 2.4847i        0                  0
        0           -1.7593 - 2.4847i        0
        0                  0             -2.4814

» [X,J]=jordan(A)
X =
0.1121             0.4440 + 0.1691i   0.4440 - 0.1691i
0.1549            -0.0775 + 0.4250i  -0.0775 - 0.4250i
0.2987            -0.1494 + 0.0434i  -0.1494 - 0.0434i

J =
-2.4814                  0                        0
     0                -1.7593                - 2.4847i
     0                   0            -1.7593 + 2.4847i
```

Forma canonică Jordan este un concept teoretic important, dar nu este indicată folosirea în cazul matricilor mari sau pentru matricile cu elemente afectate de erori de rotunjire sau de alte incertitudini.
MATLAB® -ul poate folosi în astfel de cazuri descompunerea Schur (`funcţia schur`).

Observaţie:

1. Toolbox-ul Symbolic Math extinde capacitatea MATLAB-ului prin conectarea la Maple, care este un sistem de calcul algebric performant. Una din funcţiile toolbox-ului permite calculul formei canonice Jordan.
2. Valorile proprii sunt utilizate de exemplu în studiul portretului de faze la sistemele dinamice.

7.5. Reprezentarea polinoamelor.

7.5.1. Polinoame

Polinoame

Polinoamele sunt descrise în MATLAB prin vectori linie ale căror elemente sunt de fapt coeficienţii polinoamelor în ordinea descrescătoare a puterilor.

Exemplu: Polinomul `p(x)` $=x^3+5x+6$ este reprezentat în MATLAB astfel:

```
p = [1 0 5 6]
```

Un polinom poate fi evaluat pentru o valoare a lui x cu ajutorul funcţiei **polyval**:

```
» polyval (p,1)% polinomul p în punctul x =1.
Ans =
           12
```

Se pot afla cu uşurinţă rădăcinile polinomului folosind ***funcţia roots:***

```
» r=roots(p)
r =
   0.5000 + 2.3979i
   0.5000 - 2.3979i
  -1.0000
```

Există numeroase alte funcţii şi comenzi care se ocupă cu operaţii asupra polinoamelor, funcţii care vor fi abordate într-un capitol special. Dintre acestea amintim comanda care permite înmulţirea a două polinoame, şi anume conv:

```
» p1=[1 3 5]
p1 =
     1     3     5

» p2=[2 0 1 0 5]
p2 =
     2     0     1     0     5
» p3=conv(p1,p2)
p3 =
     2     6    11     3    10    15    25
» r=17>55
r =
     0
```

MATLAB® -ul răspunde cu r = 0, adică fals.

Funcţiile polinomiale se află în directorul **polyfun:**

Funcţie	Descriere
conv	Multiplică polinoamele.
deconv	Împarte polinoamele.
poly	Returnează coeficienţii dacă se dau rădăcinile; Polinomul caracteristic.
polyder	Calcul derivatei unui polinom.
polyfit	Găsirea coeficienţilor unui polinom din aproximarea unui set de date.
polyval	Evaluarea unui polinom.
polyvalm	Evaluarea unui polinom cu argument matriceal.
residue	Descompunere în fracţii simple.
roots	Găsirea rădăcinilor unui polinom.

După cum s-a precizat deja, MATLAB® -ul reprezintă polinoamele ca vectori linie care conţin coeficienţii polinoamelor în ordinea descrescătoare a puterilor.
În continuare sunt parcurse alte câteva exemple utile.

Funcţia poly returnează coeficienţii unui polinom dacă dispunem de rădăcinile acestuia (este o funcţie inversă faţă de roots):

```
» p=[1 -1 2 4 1];
» r=roots(p)
r =
   1.0529 + 1.7248i
   1.0529 - 1.7248i
  -0.7995
  -0.3063
```

```
» coef=poly(r)
coef =
    1.0000   -1.0000    2.0000    4.0000    1.0000
```
O altă utilizare a **funcţiei poly** este aceea de calculare a coeficienţilor polinomului caracteristic al unei matrice:
```
» A
A =
    -1    -3     1
     2    -2    -1
     0     1    -3

» poly(A)
ans =
     1     6    18    23
```

Rădăcinile acestui polinom sunt chiar valorile proprii ale matricii A.
Funcţia polyval evaluează un polinom pentru o valoare specificată a argumentului.
Funcţia polyvalm permite evaluarea unui polinom în sens matriceal.
În acest caz polinomul p din exemplul anterior: $p(x) = x^4 - x^3 + 2x^2 + 4x + 1$ devine $p(X) = X^4 - X^3 + 2X^2 + 4X + I$, unde X este o matrice pătratică şi I matricea unitate.

Exemplu:
```
» C=polyvalm(p,A)
C =
   -75   -61    81
    58  -130    75
    52   -23    49
```

Funcţiile ***conv*** şi ***deconv*** implementează operaţiile de înmulţire şi împărţire a polinoamelor.

Exemple:
Fie $a(x) = x^2 + 2x + 3$ şi $b(x) = 4x^2 + 5x + 6$.

```
» a = [1 2 3]; b = [4 5 6];
» c = conv(a,b)
c =
     4    13    28    27    18
» [q,r] = deconv (c,a)
q =
     4     5     6
r =
     0     0     0     0     0
```

Funcţia ***polyder*** permite calculul derivatei unui polinom.

Exemplu:
```
» p=[1 -1 2 4 1];
» pderivat=polyder(p)
pderivat =
     4    -3     4     4
```

Funcţia polyfit găseşte coeficienţii unui polinom (o curbă) care aproximează un set de date în sensul algoritmului celor mai mici pătrate:

```
p = polyfit(x,y,n)
```

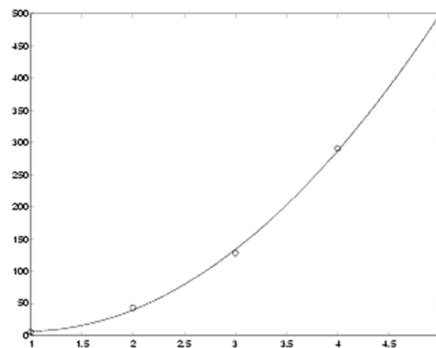
x şi y sunt vectorii care conţin setul de date iar n este ordinul polinomului ai cărui coeficienţi vor fi furnizaţi la apelarea funcţiei.

Exemplu:

```
» x = [1 2 3 4 5]; y = [5.5 43.1 128 290.7 498.4];
» p = polyfit(x,y,3)
p =
   -0.1917   31.5821  -60.3262   35.3400
```

Pentru plotarea rezultatului se utilizează mai întâi funcţia polyval pentru o trasare cât mai exactă a graficului polinomului şi apoi se reprezintă graphic estimarea versus datele reale pentru eventuale comparaţii.

```
» x2 = 1:.1:5;
» y2 = polyval(p,x2);
» plot(x,y,'o',x2,y2)
```



Funcţia residue se utilizează pentru descompunerea în fracţii simple. Se aplică în cazul raportului a două polinoame b(x) şi a(x).

Exemplu:

$$\frac{-4 + 8/s}{1 + 6/s + 8/s\text{^}2}$$

```
» b = [-4 8];
» a = [1 6 8];
»  [r,p,k] = residue(b,a)
r =
   -12
     8
p =
    -4
    -2
k =
     [ ]
```

Dacă se folosesc trei argumente de intrare (r, p, şi k), funcţia residue asigură conversia înapoi în forma polinomială:

```
»  [b2,a2] = residue(r,p,k)
b2 =
    -4      8
a2 =
     1      6      8
```

7.5.2. Interpolarea

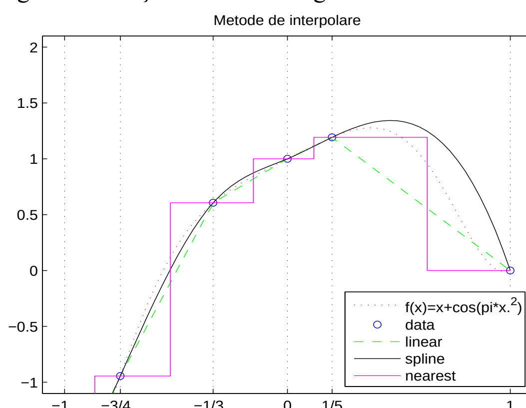
Interpolarea este un proces de estimare a valorilor dintre date (puncte) cunoscute.
Aplicaţiile interpolării sunt numeroase în domenii cum ar fi procesarea numerică a semnalelor şi imaginilor , rezolvarea ecuaţiilor diferenţiale şi cu derivate parţiale.
MATLAB® -ul dispune de mai multe tehnici de interpolare, (29 ,pp. 209:215) alegerea unei metode sau alteia făcându-se în funcţie de acurateţea necesară, de viteza de execuţie şi de gradul de utilizare a memoriei.
Funcţiile de interpolare se află în directorul **polyfun**.

Funcţie	Descriere
linear	Linear interpolation.
interp1	Mono-dimensional interpolation
interp2	Bi-dimensional interpolation.
interp3	Tri-dimensional interpolation.
nearest	Nearest nighbor interpolation.
spline	B-spline cubic interpolation

Exemplu
Alegem cinci puncte de pe graficul funcţiei `f(x)=x+sin π*x^2.`
Vom calcula un interpolant folosind metodele `nearest,linear, cubic-spline.`

Codul Matlab este:
```
x=[-1,-3/4,-1/3,0,1/5,1];
y=x+cos(pi*x.^2);
xi=linspace(-1,1,60);
yi=xi+cos(pi*xi.^2);
yn=interp1(x,y,xi,'nearest');
y1=interp1(x,y,xi,'linear');
ys=interp1(x,y,xi,'spline');
plot(xi,yi,'r:',x,y,'bo','MarkerSize',5); hold on
plot(xi,y1,'g--',xi,ys,'k-')
stairs(xi,yn,'m-')
set(gca,'XTick',x);
set(gca,'XTickLabel','-1|-3/4|-1/3|0|1/5|1')
set(gca,'XGrid','on')
axis([-1.1,1.1,-1.1,2.1])
legend('f(x)=x+cos(pi*x.^2)','data','linear','spline','nearest',4)
title('Metode de interpolare')
hold off
```

În urma execuţiei programului obţinem următorul grafic:



7.5.3. Interpolarea prin metoda transformatei Fourier

Funcţia `interpft` interpolează datele cu o singură variabilă utilizând metoda FFT (Fast Fourier Transform) care se apelează cu sintaxa:

```
y=interpft (x,n)
```

care returnează un vector `y` de lungime `n` obţinut din vectorul `x`. Numărul `n` trebuie să fie mai mare decât numărul de elemente ale vectorului `x`, iar rezultatul are periodicitatea `circulară` dată de utilizarea transformatei Fourier.

Concluzie: Diferenţa maximă înre valorile interpolate şi cele reale este egală cu ordinul de mărime al celui mai mic număr reprezentabil în calculator.

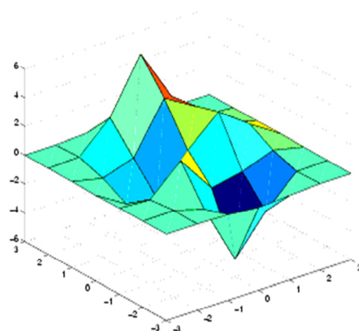
7.5.4 Compararea unor metode de interpolare bi-dimensională

Exemplu.

Folosirea unor metode de interpolare bi-dimensionala pentru o matrice de date *7 x 7.*

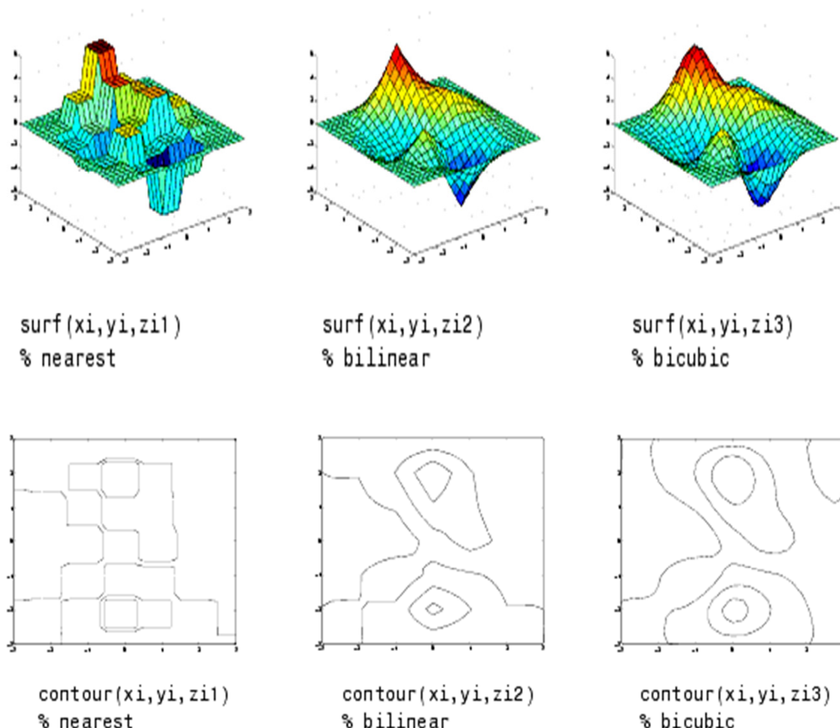
Generarea funcţiei peaks (cu rezoluţie mică) se face folosind următoarele comenzi:

```
[x,y] = meshgrid(-3:1:3);
z = peaks(x,y);
Surf(x,y,z);
```



```
Generarea unei suprafeţe mesh fine pentru interpolare:
[xi,yi] = meshgrid(-3:0.25:3);
Interpolarea cu metoda celei mai apropiate vecinătăţi:
zi1 = interp2(x,y,z,xi,yi,'nearest');
Interpolarea cu metoda biliniară:
zi2 = interp2(x,y,z,xi,yi,'bilinear');
Interpolarea cu metoda bicubică:
zi3 = interp2(x,y,z,xi,yi,'bicubic');
```

Compararea graficelor corespunzătoare precum şi a contururilor suprafeţelor în cazul diferitelor metode de interpolare sunt prezenatate în figurile de mai jos:



Se observă că metoda bicubică produce cele mai netede contururi. O metodă cum ar fi cea a celor mai apropiate vecinătăţi este preferată însă în anumite aplicaţii, cum ar fi cele medicale unde nu trebuie generate date noi.

7.5.5. Aproximarea prin metoda celor mai mici pătrate

Ideea este de an aproxima un set de date printr-o linie dreaptă şi apoi aproximarea printr-un polinom. Pentru ca aproximarea să fie considerată ***cea mai bună*** suma pătratelor distanţelor de la fiecare punct la curba aproximantă (linie sau polinom) trebuie să fie minimă [4].

Cu această condiţie este posibil ca nici un punct al setului de date să nu se găsească pe curba aproximantă. Acest lucru separă net aproximarea de interpolare, la care toate punctele sunt situate pe curbă.

Regresia liniară

Este o metodă de aproximare unui set de date printr-o dependenţă liniară care minimizează suma pătratelor dintre dreapta de aproximare şi punctele date. Măsura calităţii unei aproximări liniare este dată de suma pătratelor distanţelor de la fiecare punct la estimaţia liniară care se poate evalua astfel:

```
sum_p=sum((y-y1).^2)
```

Determinarea parametrilor m şi n ai dreptei de aproximare `y = mx+n` se face utilizând funcţia `polyfit`.

Regresia polinomială

Este o metodă de aproximare a unui set de date printr-un polinom de grad N. Dacă setul de date are N elemente, toate datele se află pe curba de aproximare. Pentru un grad al polinomului mai mic decât numărul de date, aproximarea este cu atât mai bună cu cât gradul polinomului este mai apropiat de numărul de date. Utilizarea unui polinom de aproximare cu grad mai mare decât setul de date poate conduce la erori de aproximare considerabile. Determinarea celei mai bune aproximări a unui set de date (x,y) cu un polinom de `ordin n` se foloseşte funcţia `polyfit,` care se apelează cu sintaxa :

```
p=polyfit(x,y,n)
```

Funcţia `polyfit` returnează coeficienţii polinomului p(x) care în punctele precizate de vectorul x are, în sensul celor mai mici pătrate , valorile date de vectorul y.

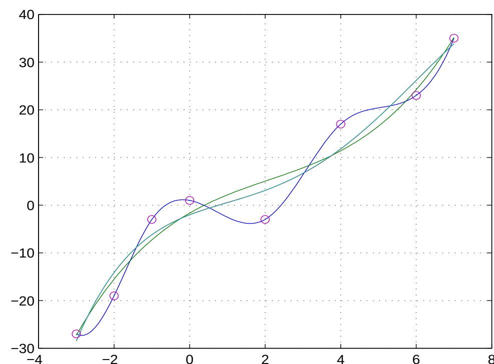
Exemplu

Să se scrie o secvenţă de program care să aproximeze prin polinoame de grad 3, 4, 5 şi prin funcţii B- spline 5 perechi de puncte de coordonate cunoscute. Să se reprezinte grafic aproximările determinate.

Cu secvenţa **de program Matlab**:

```
close all
clear all
x=[-3 -2 -1 0 2 4 6 7];
y=[-27 -19 -3 1 -3 17 23 35];
c2=polyfit(x,y,3);% coef. polinom. de grad 3
c5=polyfit(x,y,4);% coef polinom. de grad 4
c7=polyfit(x,y,5);% coef polinom. de grad 5
x1=-3:.1:7;
y2=polyval(c2,x1);% evaluarea polinomului de grad 3
y5=polyval(c5,x1);% evaluarea polinomului de grad 4
y7=polyval(c7,x1);% evaluarea polinomului de grad 5
yi=spline(x,y,x1);% interpoleaza spline
plot(x1,yi,x1,y2,x1,y5,x1,y7,x,y,'o')
grid on
```

se obţine graficul următor:



7.6. Rezolvarea ecuaţiilor neliniare

Reamintim câteva metode analitice de rezolvare a ecuaţiilor neliniare:

- Metoda lui *Newton*
- Metoda lui *Broyden*
- Metoda Secantei
- Metoda Falsei Poziţii
- Metoda Punctului fix

Sistemul Matlab pune la dispoziţia utilizatorilor o funcţie care permite calculul zerourilor ecuaţiilor neliniare.

Aceasta este ***fzero***, care se apelează astfel:

`[x,fval]=fzero ('fun',[a,b],optimset)` unde

```
'fun'= funcţia neliniară
[a,b]= este intervalul în care se caută soluţia aproximativă
optimset function = funcţie care controlează rezolvarea acestor
ecuaţii astfel:
dacă display option   = ,off',nu generează nici un raport
                      ='iter',la fiecare iteraţie afişează un raport
                      = ,final',afişiează un raport doar la sfârşitul
execuţiei
                      ='notify',afişează un raport final dacă metoda
este divergentă (opţiune implicită)
```

Tolx option = toleranţa impusă pentru convergenţă

FunValCheck = determină şi soluţii complexe

Exemplu

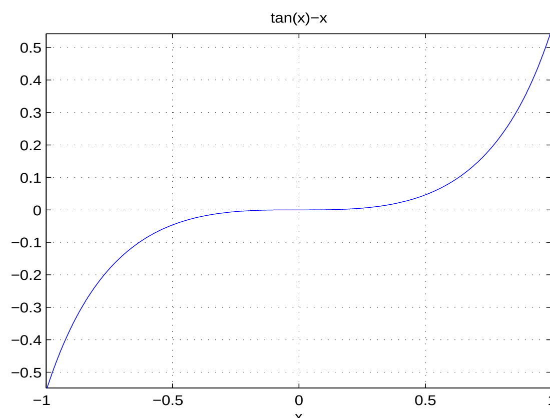
```
>> os = optimset('Display','Final');
>> [x,fval]=fzero('tan(x)-x',[-1,1],os);
Zero found in interval:[-1,1].
x =0
```

fval=$\frac{0}{0}$

Dacă în linia de comandă tastăm:

```
>> ezplot('tan(x)-x',[-1,1]),grid
```

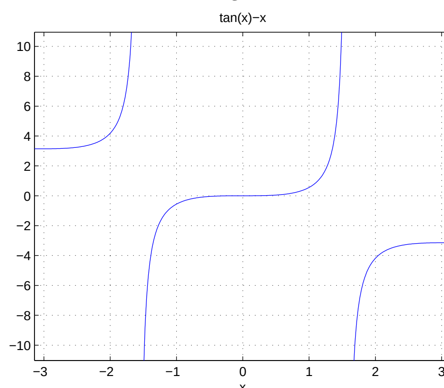
Se obţine graficul:



Dacă în linia de comandă tastăm:

```
>> ezplot('tan(x)-x',[-pi,pi]),grid
```

Şi deoarece funcţia `f(x)=tan (x)` are singularităţi în ± `pi/2` se obţine graficul:



Deci în acest caz este indicat de a găsi soluţiile unor astfel de ecuaţii scriind funcţii proprii.

Exemplu [29, p. 294]

Folosind metoda lui *Newton* să se rezolve sistemul neliniar:

$$3x_1-\cos(x_1x_2)-\frac{1}{2}=0,$$

$$x_1^2-81(x_2+0.1)+\sin x_3+1.06=0$$

$$e^{-x1x2}+20x_3+\frac{10\pi-3}{3}=0$$

Vom încerca să scriem o funcţie proprie pentru aceasta metodă astfel:

```
function [y,ni]=Newton (f,fd,x0,ea,er,nmax)
%Metoda lui Newton
%apelul se face astfel: call[z,ni]=Newton(f,fd,x0,ea,er,nmax)
%Intrări sunt: f-funcţia
%fd-derivata
%x0-valoarea de start
%ea,er-eroarea absolută si relativă
%nmax-numărul maxim de iteraţii
%Iesiri
```

```
%z- soluţia aproximativă
%ni-numărul curent de iteraţii
If nargin < 6, nmax=50;end
If nargin < 5, err=0;end
If nargin < 4, ea=1e-3;end
xp=x0(:);% valoarea obţinută la iteraţia anterioară
for k=1:nmax
xc=xp-fd(xp)\f(xp);
    if norm(xc-xp,inf)<ea+er*norm(xc,inf)
           z=xc;%succes
           ni=k;
           return
           end
           xp=xc;
    end
error('s-a depasit numarul maxim de iteraţii ')
```

Apoi trebuie scrisă funcţia care conţine ecuaţiile sistemului neliniar care se mai numeşte şi **câmpul vectorial.**

```
function y=ecs3(x)
%aici se dau ecuaţiile neliniare ale sistemului
y=[3*x(1)-cos(x(2)*x(3))-1/2;…
x(1)^2-81*(x(2)+0.1)^2+sin(x(3))+1.06;…
exp(-x(1)*x(2))+20*x(3)+(10*pi-3)/3];
```

şi Jacobianul în Matlab:

```
function y=decs3d(x)
y=[3,x(3)*sin(x(2)*x(3)),x(2)*sin(x(2)*x(3));…
2*x(1),-162*(x(2)+0.1),cos(x(3));…
-x(2)*exp(-x(1)*x(2)),-x(1)*exp(-x(1)*x(2)),20];
```

Apelul pentru o valoare de start x^0= [0.1,0.1,-0.1]^T se face astfel:

```
>> [z,ni]=Newton(@ecs3,@decs3,x0,1e-9)
```

CAPITOLUL 8.

ANALIZĂ MATEMATICĂ

8.1. Reprezentarea grafică a funcţiilor matematice

Funcţiile matematice uzuale sunt furnizate de MATLAB® ca funcţii built-in (cum ar fi `sin, cos, log10, log, atan` etc.).
Pentru reprezentarea altor funcţii matematice se utilizează exprimarea în fişiere tip .m .
De exemplu, o funcţie cum este următoarea:

$$f(x)=1/((x-0.3)^2 +0.01)+ 1/((x-0.9)^2+0.04)-6$$

poate fi creată într-un fişier MATLAB de **tip function** şi poate fi utilizată ulterior ca intrare în alte funcţii (aşa-numitele funcţii de funcţii).

```
function y = nelin(x)
y = 1./((x-0.3).^2+0.01)+1./((x-0.9).^2+0.04)-6;
```

O altă posibilitate este crearea la nivelul liniei de comandă a unui **obiect inline** prin folosirea unei expresii tip şir de caractere:

```
» f=inline(`1./((x-.3).^2+.01)+1./((x-.9).^2+.04)-6');
```

Pentru a evalua această funcţie f în 2.0 tastăm simplu:

```
» f(2.0)

ans =
   -4.8552
```

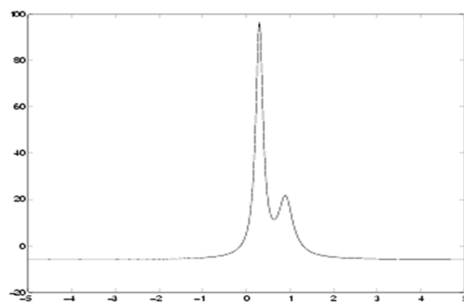
Alt exemplu:

```
» f = inline('y*sin(x)+x*cos(y)','x','y')
» f(pi,2*pi)
ans =
     3.1416
```

8.1.1. Reprezentarea grafică a funcţiilor

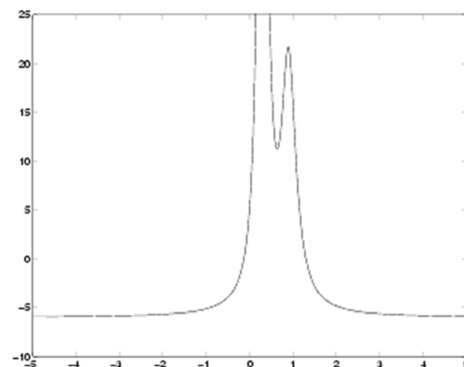
Pentru reprezentarea grafică a funcţiilor se poate utiliza funcţia `fplot`. Se pot controla limitele axelor de reprezentare grafică.
Exemplu: Trasarea graficului funcţiei nelin pentru limitele [-5 5] ale axei x:

```
fplot('nelin',[-5 5])
```



Dacă dorim şi precizarea limitelor de reprezentare pe axa y (realizarea unui zoom) folosim comanda:

```
fplot('nelin',[-5 5 -10 25])
```

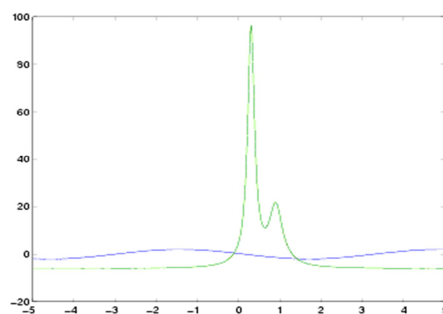


Un alt exemplu de folosire directă a funcţiei fplot:

```
fplot('2*sin(x+3)',[-1 1]);
```

Se poate realiza şi reprezentarea mai multor funcţii pe acelaşi grafic:

```
fplot('[2*sin(x+3), nelin(x)]',[-1 1]);
```



Exemplu [6]

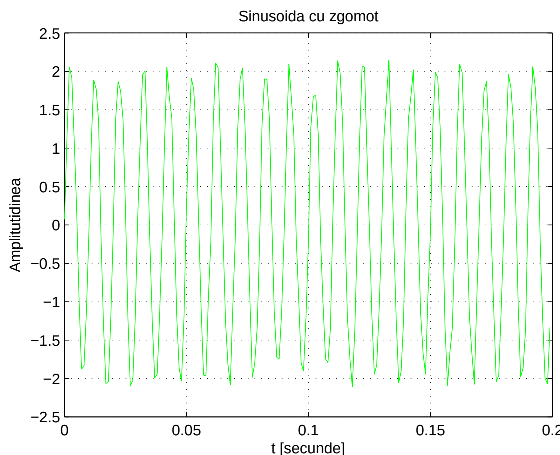
Să se scrie un program în MATLAB care sa reprezinte grafic un semnal sinosoidal cu amplitudinea A = 2V, frecventa f=100Hz,faza initiala fi=0, esantionat cu frecventa fe = 1000hz, peste care ,peste care se suprapune un zgomot uniform distribuit [-0.25, 0.25]V.

Să se calculeze raportul semnal/zgomot.

Codul Matlab

```
fe=1000;
f=100;
a=0.25;
N=200;
A=2;
RSZdB=10*log10((A^2/2)/((2*a)^2/12));
t=(0:N-1)*1/fe;
s=A*sin(2*pi*f*t)+(2*a*rand(size(t))-a);
plot(t,s,'g-')
title('Sinusoida cu zgomot');
xlabel('t [secunde]');
ylabel('Amplitutidinea');
grid;
```

În urma execuţiei acestui exemplu rezultă graficul:



8.2. Minimizarea funcţiilor şi găsirea zerourilor

MATLAB® -ul furnizează o serie de funcţii de nivel înalt care realizează sarcini de optimizare a funcţiilor. Aceste funcţii sunt grupate în principal pe următoarele domenii:

- Minimizarea funcţiilor de o variabilă
- Minimizarea funcţiilor de mai multe variabile
- Setarea opţiunilor de minimizare
- Găsirea zerourilor unor funcţii

Pentru exemplificare vom considera minimizarea unei funcţii de o singură variabilă şi găsirea zerourilor pentru această funcţie.

8.2.1. Minimizarea unei funcţii

Pentru găsirea unui minim local al unei funcţii scrise într-un fişier function , se utilizează funcţia ***fminbnd.***
Pentru găsirea minimului funcţiei `nelin` în intervalul (0.3, 1) folosim instrucţiunea:

```
» x = fminbnd('nelin',0.3,1)
x =
     0.6370
```

Dacă dorim o afişare detaliată a paşilor făcuţi de prodedura de minimizare se utilizează următoarea sintaxă:

```
» x = fminbnd('nelin',0.3,1,optimset('Display','iter'))

Func-count       x            f(x)          Procedure
    1        0.567376      12.9098         initial
    2        0.732624      13.7746         golden
    3        0.465248      25.1714         golden
    4        0.644416      11.2693         parabolic
    5        0.6413        11.2583         parabolic
    6        0.637618      11.2529         parabolic
    7        0.636985      11.2528         parabolic
    8        0.637019      11.2528         parabolic
    9        0.637052      11.2528         parabolic

x =
     0.6370
```

8.2.2. Găsirea zerourilor

Funcţia `fzero` permite găsirea zerourilor unei funcţii (este vorba de fapt de o ecuaţie cu o singură necunoscută).
Dacă se dă un punct de plecare x0 pentru procedura de căutare, fzero va căuta un interval în jurul acestui punct în care funcţia schimbă semnul. Dacă un astfel de interval este găsit, fzero returnează valoarea pentru care funcţia schimbă semnul (adică zeroul), iar dacă un astfel de interval nu este găsit returnează NaN.
Altă variantă permite căutarea într-un interval specificat de utilizator.

Exemplu: Găsirea unui zerou al funcţiei ***nelin*** în apropiere de -0.2:

```
» a = fzero('nelin',-0.2)
a =
-0.1316
```

Pentru verificare evaluăm funcţia în punctul -0.1316 şi găsim într-adevăr o valoare foarte aproape de zero:

```
» nelin(a)
ans =
   8.8818e -16
```

Se poate folosi şi varianta cu interval de căutare precizat de utilizator.

În continuare este prezentată această variantă plus folosirea unor opţiuni suplimentare pentru afişarea detaliată a informaţiilor despre procedură şi a paşilor:

```
» options = optimset('Display','iter');
» a = fzero('nelin',[-1 1],options)
Func-count      x           f(x)          Procedure
    1          -1          -5.13779           initial
    2           1            16               initial
    3    -0.513876      -4.02235              interpolation
    4     0.243062       71.6382              bisection
    5    -0.473635      -3.83767              interpolation
    6    -0.115287      0.414441              bisection
    7    -0.150214     -0.423446              interpolation
    8    -0.132562    -0.0226907              interpolation
    9    -0.131666    -0.0011492              interpolation
   10    -0.131618  1.88371e-007              interpolation
   11    -0.131618  -2.7935e-011              interpoation
   12    -0.131618  8.88178e-016              interpolation
   13    -0.131618 -9.76996e-015        interpolation
Zero found in the interval: [-1, 1].
a =
   -0.1316
```

8.3. Integrarea şi derivarea numerică a funcţiilor

Integrarea şi derivarea sunt concepte fundamentale în rezolvarea unui număr mare de probleme în inginerie şi în ştiinţă. În multe situaţii nu se pot da soluţii analitice, deci este nevoie de aplicare a unor metode de integrare şi derivare numerică .

8.3.1. Integrarea numerică

Aria subgraficului unei funcţii f(x) poate fi determinată prin integrarea numerică, proces care se numeşte quadratură (quadrature). Pentru rezolvarea integrării numerice în cazul monodimensional există următoarele funcţii MATLAB:

- `quad`, care foloseşte un algoritm de tip Simpson
- `quad8`, care utilizează un algoritm de tip Newton
- `trapez`, calculează integrala prin metoda trapezelor.

Observaţie

Funcţiile `quad`, `quad8` se utilizează, dacă integrantul este exprimat sub forma unei funcţii analitice.

Funcţia `trapez`, presupune că integrantul este dat prin valori numerice în noduri echidistante ale intervalului de integrare.

Funcţiile `quad`, `quad8` pot trata unele singularităţi care se găsesc la unul din capete, însă nu rezolvă singularităţi în interiorul intervalului.

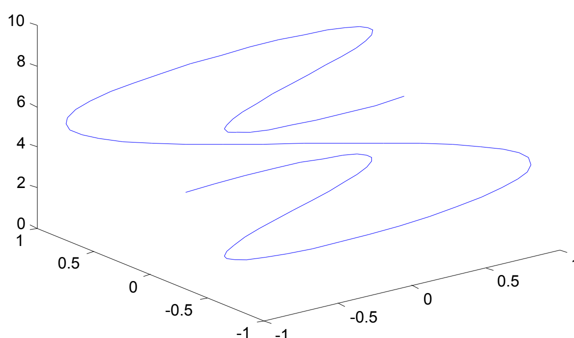
Exemplu: Pentru integrarea funcţiei nelin între 0 şi 1 folosim comanda

```
» q = quad('nelin',0,1)
q =
    29.8583
```

Funcţiile ***quad*** sau ***quad8*** permit şi alte argumente de intrare care specifică eroarea tolerată pentru integrare şi alte opţiuni (a se vedea cu help).

Exemplu de integrare numerică:
Calculul lungimii unei curbe
Vom considera o curbă dată de ecuaţiile:
`x(t)=sin(2t), y(t)= cos(t), z(t)=t,`
`cu: t∈ [0,3π]`
O plotare tridimensională a acestei curbe poate fi obţinută cu

```
» t = 0:0.1:3*pi;
» plot3(sin(2*t),cos(t),t)
```



Lungimea acestei curbe este dată de formula următoare:

$$\int_0^{3\pi} (\sqrt{4\cos(2t)^2 + \sin(t)^2} + 1)dt$$

Pentru calculul lungimii trebuie integrată numeric integrala de mai sus. Pentru aceasta se creează mai întâi o funcţie MATLAB® care descrie integrantul pe care o numim `fcurba`:

```
function f = fcurba(t)
f = sqrt(4*cos(2*t).^2 + sin(t).^2 + 1);
şi apoi se integrează cu ajutorul funcţiei quad:
lungime = quad('fc',0,3*pi)
lungime =
   1.7222e+01
```

Exemplu

```
%Calculul integralelor cu metoda trapezelor
function z=trapez
x=.5:0.1:2.5; %limita inferioara: pasul: limita superioara
y=sin(x.^2); % integrandul
z=0.1*trapz(y); % se inmulteste cu pasul, implicit pasul fiind 1
disp('Valoarea aproximativa a integralei');
```

Apelarea se face scriind *trapez*, iar valoarea afişată este 0.3924 .

8.3.2. Derivarea numerică

Derivarea unei funcţii f(x) este viteza de variaţie a funcţiei în raport cu variabila x. Interpretarea geometrică a derivatei într-un punct este panta tangentei la graficul funcţiei în punctual considerat. Punctele cu derivata nulă (puncte critice) pot reprezenta fie o regiune orizontală a funcţiei, fie un punct de maxim, sau un punct de minim local al funcţiei.
Dacă derivata a doua în punctul critic este pozitivă, atunci valoarea funcţiei în punctual respective este un minim local, iar dacă derivata a doua a punctului critic este negativă, atunci valoarea funcţiei în acel punct este un maxim local. De regulă derivarea numeric amplifică micile erori, uneori îndepărtându-se semnificativ de la soluţia analitică.
Funcţia `diff` permite aproximarea derivatei numerice a unei funcţii y(x) cu relaţia:

```
dy=diff(y)./diff(x)
```

Exemplu

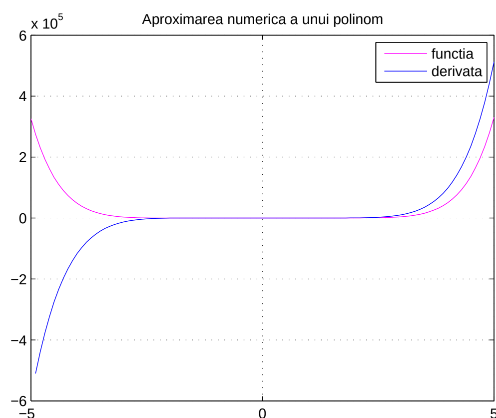
Să se calculeze derivata funcţiei:

$$f(x)=x^8-4x^6+16x^3+2x^2+3x-11$$

pe intervalul [-5,5] utilizând aproximarea cu diferenţe progressive.
Programul MATLAB este:

```
x=-5:0.1:5;
f=x.^8-4*x.^6+16*x.^3+2*x.^2+3*x-11;
df=diff(f)./diff(x);
xd=x(2:length (x));
plot(x,f,'m-');
hold on
plot(xd,df,'b-')
title('Aproximarea numerica a unui polinom');
legend('functia','derivata');
grid;
```

Se obţine următorul grafic:



Observaţie

Folosind funcţia `diff` se poate efectua derivarea numerică bazată pe polinomul de interpolare **Newton.**

8.3.3. Aproximarea numerică a laplaceanului şi a gradientului

Funcţia MATLAB® `del2` calculează Laplaceanul discret în cinci puncte. Se apelează cu sintaxa :

```
V=del2(U);
```

Funcţiile Matlab :

```
gradient -aproximeaza gradientul unei funcţii
quiver- reprezinta orientarea unui câmp de vectori
```

Exemplu

Să se reprezinte grafic gradientul unui câmp de vectori asociat funcţiei:

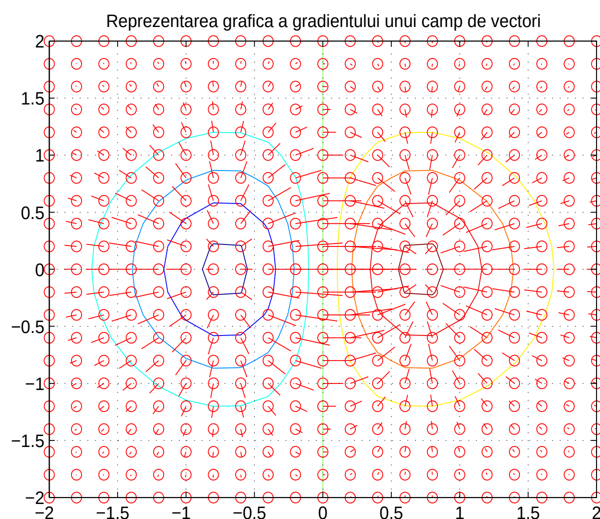
$$z(x,y)=x*e^{-x*x-y*y}$$

în domeniul [-2,2] X [-2,2].

Programul Matlab:

```
% Gradient
xp=-2:.2:2;
yp=-2:.2:2;
[x,y]=meshgrid(xp,yp);
z = x.*exp(-x.^2-y.^2);
[px,py] = gradient(z,.2,.2);
contour(x,y,z); hold on;
quiver(x,y,px,py,1.5,'go');
title('Reprezentarea grafica a gradientului unui camp de vectori');
grid on
hold off
```

Se obţine următoarea reprezentare grafică:



8.4. Funcţii speciale

8.4.1. Ecuaţia şi funcţiile Bessel

Ecuaţia diferenţială:

$$x^2y''(x) + xy' + (x^2-\nu^2)y(x)=0, \text{ pentru } x > 0 \text{ şi } \nu=\text{constant} \geq 0.$$

este ecuaţia lui *Bessel* . Această ecuaţie apare în rezolvarea unor probleme de câmp cu simetrie cilindrică.
Soluţia generală a acestei ecuaţii diferenţiale pentru $\nu \notin Z$ este :

$$y(x)=C_1J_\nu(x)+C_2\ J_{-\nu}(x)$$

unde C_1, C_2 sunt constante reale este funcţia *Bessel de speţa I* şi ordin ν.
Soluţia generală a ecuaţiei *Bessel* pentru ν=n=întreg este :

$$y(x)=C_1J_\nu(x)+C_2\ Y_n(x)$$

unde C_1, C_2 sunt constante reale, iar $Y_n(x)$ este funcţia *Bessel de speţa II* şi de ordin n.
Ecuaţia diferenţială:

$$x^2y''(x) + xy' - (x^2+\nu^2)y(x)=0, \text{ pentru } x > 0 \text{ şi } \nu=\text{constant şi pozitiv}.$$

este cunoscută sub numele de ecuaţia lui *Bessel* modificată.
Soluţia generală a acestei ecuaţii diferenţiale pentru $\nu \notin Z$ este :

$$y(x)=C_1I_\nu(x)+C_2\ I_{-\nu}(x)$$

unde C_1, C_2 sunt constante reale, iar I_ν este funcţia *Bessel* modificată de speţa I şi ordin ν .
Soluţia generală a ecuaţiei *Bessel* pentru ν=n=întreg este :

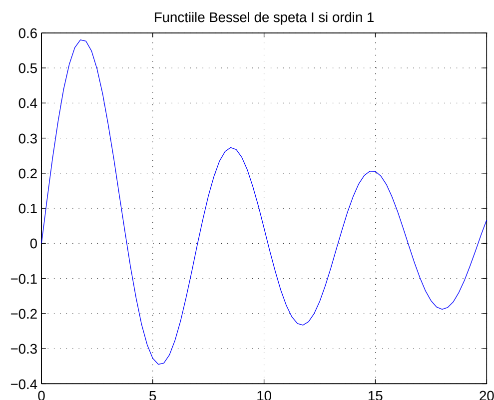
$$y(x)=C_1I_n(x)+C_2\ K_n(x)$$

unde C_1, C_2 sunt constante reale, iar I_n şi K_n sunt funcţiile Bessel modificate de speţa I şi II şi ordin n.
Sistemul MATLAB® pune la dispoziţie următoarele funcţii:
Besselj=calculează funcţiile Bessel de speţa I
Bessely=calculează funcţiile Bessel de speţa II
Besseli=calculează funcţiile Bessel modificate de speţa I
Besselk=calculează funcţiile Bessel modificate de speţa II.

Exemplu
Să se reprezinte grafic funcţiile *Bessel* de speţa I şi de ordin 1.
În urma executării următorului program Matlab :

```
x=0:.25:20;
J=besselj(1,x);
plot(x,J);
grid
title('Functiile Bessel de speta I')
```

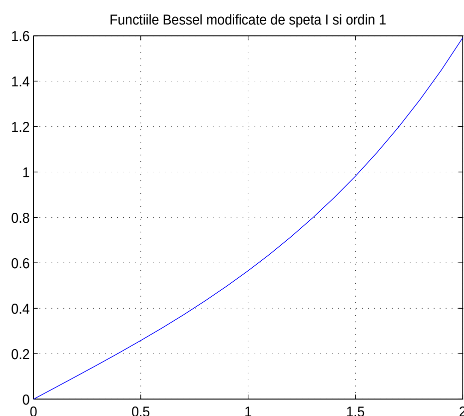
rezultă graficul:



Exemplu
Să se reprezinte grafic funcţiile *Bessel* modificate de speţa I şi ordin 1.
În urma executării programului

```
x=0:.1:2;
N=besseli(1,x);
plot(x,N);
grid
title('Functiile Bessel modificate de speta I si ordin 1')
```

rezultă graficul:



8.4.2. Funcţia Gamma

Funcţia *Gamma* este definită de integrala:

$$\Gamma(a) = \int_0^\infty t^{a-1} e^{-t} dt \ , \quad a \in R$$

Dacă argumentul a este întreg , funcţia Gamma este identică cu factorialul.
Funcţia Matlab care calculează funcţia $\Gamma(a)$ se apelează cu sintaxa:

```
G=gamma(a);
```

Funcţia *Gamma* incompletă :

$$P(x,a) = \frac{1}{\Gamma(a)} \int_0^x \int_0^\infty t^{a-1} e^{-t} dt \qquad x, a \in R, \ a > 0.$$

Se apelează cu funcţia Matlab:

```
P = gammainc(x,a)
```

Argumentul x trebuie să fie real (scalar sau vector), iar a trebuie să fie scalar real şi pozitiv. Funcţia `gammaln` returnează logaritmul natural din valorile lui lui `gamma,` se apelează cu sintaxa :

```
Gl=gammaln(a);
```

8.4.3. Funcţia Beta

Funcţia `Beta` este descrisă de expresia:

$$B(z,w) = B(w,z) = \frac{\Gamma(z)\,\Gamma(w)}{\Gamma(z+a)}$$

Se apelează cu sintaxa :

```
B = beta(z,w)
```

iar funcţia `Beta` incomplectă se apelează cu sintaxa:

```
I =betainc(x,z,w)
```

Funcţia `Matlab betaln` returnează logaritmul natural din valorile lui `beta` şi se apelează cu sintaxa:

```
Bl=betaln(z,w)
```

Variabilele `z` şi `w` pot fi vectori sau matrice de aceleaşi dimensiune.

Exemplu:
Dacă tastăm în linia de comanda:

```
>> format rat
>> beta((0:10)',3)
```

Se obţine rezultatul:

```
ans =
       1/0
       1/3
       1/12
       1/30
       1/60
       1/105
       1/168
       1/252
       1/360
       1/495
       1/660
```

8.4.4. Funcţia eroare

Funcţia eroare `erf` şi complementara funcţiei eroare sunt cazuri speciale ale funcţiei *Gamma* incomplecte şi sunt definite de relaţiile:

$$\mathrm{Erf}(x) = \frac{2}{\sqrt{\pi}}\int_0^x e^{-t*t}\,dt$$

şi

$$\mathrm{Erfc}(x) = \frac{2}{\sqrt{\pi}}\int_x^\infty e^{-t*t}\,dt$$

Relaţia dintre funcţia `eroare` şi funcţia `Gamma incompletă` este:

$$Erf(x)=P(x^2,0.5),\ x \geq 0.$$

Funcţiile `erf` şi `erfc` se apelează cu sintaxele:

```
y=erf(x), y=erfc(x);
```

Inversa funcţiei eroare, `erfinv`, returnează valoarea lui `x` pentru valori cunoscute ale lui y şi se apelează cu sintaxa:

```
x=erfinv(y).
```

Valorile lui y trebuie să fie în intervalul [-1 1]$\Rightarrow$ x $\in$ (-∞, ∞).
Pentru a calcula valoarea funcţiei eroare generalizată se utilizează secvenţa:

```
E=erf(y)-erf(x)
```

Argumentul x poate fi scalar, vector sau matrice, rezultatul fiind de aceleaşi dimensiuni.

8.4.5. Funcţia de repartiţie Laplace

Funcţia densităţii de probabilitate a repartiţiei normale normate este definită de relaţia:

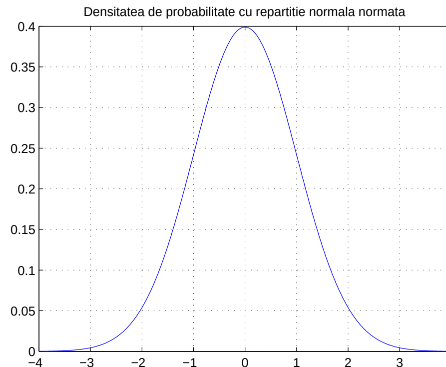
$$f(z)=\frac{1}{\sqrt{2\pi}}\ e^{-\frac{z^2}{2}}$$

Exemplu

Dacă $z\in[-4,4]$, $f(z)$ reprezintă densitatea de probabilitate cu repartiţie normală pentru care utilizăm secvenţa Matlab:

```
z=-4:0.01:4;
f=1./(sqrt(2*pi))*exp(-z.^2/2);
plot(z,f); grid on
title('Densitatea de probabilitate cu repartitie normala normata')
```

se obţine graficul:



Pentru calculul funcţiei de repartiţie *Laplace* se foloseşte secvenţa Matlab:

```
F=trapz(z, 1./(sqrt(2*pi))*exp(-z.^2/2);
```

Limita de integrare -∞ se va lua egală cu un număr între -5 şi -10, fără ca prin aceasta să se introducă erori semnificative.

CAPITOLUL 9.

ELEMENTE DE TEORIA PROBABILITĂŢILOR ŞI STATISTICĂ MATEMATICĂ

9.1. Legi de probabilitate de tip discret

Menţionez că următoarele definiţii ,teoreme, proprietăţi şi unele din programe sunt preluate din lucrarea [1] cu acceptul autorului.

Definiţie. Dacă variabila aleatoare X este de tip discret, adică are un număr cel mult numărabil de valori, fie acestea $x_i \in R$, $i \in I$, atunci funcţia de repartiţie F ataşată este o funcţie în scară şi este dată prin:

$$F(x)=\sum_{xi \le x}^{i \in I} pi, \ \forall x \in R$$

unde $p_i = P(X=x_i)$.

Definiţie. Numim ***distribuţia sau repartiţia variabilei aleatoare X*** de tip discret tabloul: $X=(x_i,p_i)$ unde $x_i \in R$, $i \in I$, sunt valorile pe care le ia variabila aleatoare X ia valoarea x_i, adică $p_i = P(X=x_i)$, pentru fiecare $i \in I$. Evenimentele $(X=x_i)$, formează un sistem complet de evenimente, dacă $\sum_{i \in I} pi = 1$.

Forma cea mai generală a unei variabile aleatoare aparţinând unei clase se numeşte *lege de probabilitate de tip discret.*

În MATLAB® distribuţia variabilei aleatoare X poate fi precizată prin funcţia de probabilitate `pdf`, sau prin funcţia de repartiţie `cdf`.

Funcţia pdf

Două moduri de apel pentru calculul valorilor funcţiei `pdf` avem :

```
P=pdf('legea',x,par1,par2,…)
P=numef(x,par1,par2,…)
```

`unde legea` este un şir de caractere predefinit pentru fiecare din legile de probabilitate definite în `Statistic toolbox`, `numef` este un un şir de caractere din care ultimele trei sunt `pdf` iar cele ce le preced sunt cele care dau numele predefinit al legii de probabilitate (ca şi cele din parametrul `legea`).

În urma executării uneia din cele două instrucţiuni, se calculează matricea `p` a probabilităţilor legii precizată prin parametrii `legea` respectiv `numef` corespunzătoare valorilor date prin matricea X şi având parametrii daţi prin matricele: `par1,par2, ...` Aceste matrici trebuie să fie de aceleaşi dimensiuni, cu excepţia că dacă unele din aceştia sunt scalari, caz în care aceştia se extind la matricele constante de aceleaşi dimensiuni cu celelate şi care iau valorile scalarilor corespunzători.

Funcţia cdf

Există două forme de apel:

```
Cp=cdf('legea',x,par1,par2,…)
Cp= numef(x,par1,par2,…)
```

`unde legea` este un şir de caractere predefinit pentru fiecare din legile de probabilitate definite în `Statistic toolbox`, `numef` este un un şir de caractere din care ultimele trei sunt `cdf`, iar cele ce le preced sunt cele care dau numele predefinit al legii de probabilitate (ca şi cele din parametrul `legea`).

În urma executării uneia din cele două instrucţiuni, se calculează matricea `cp` a probabilităţilor cumulate legii precizată prin parametrii `legea` respectiv `numef` corespunzătoare valorilor date prin matricea X şi având parametrii daţi prin `matricele: par1,par2,...`Aceste matrici trebuie să fie de aceleaşi dimensiuni, cu excepţia că dacă unele din aceştia sunt scalari, aceştia se extind la matricele constante de aceleaşi dimensiuni cu celelate şi care iau valorile scalarilor corespunzători.

Următoarele legi de probabilitate de tip discret au implementate funcţii Matlab:

Legea lui Bernoulli, Legea uniformă discretă (`unid`,`Legea binomială(bino)`, `Legea hypergeometrică (hyge`, `Legea lui Poisson (poiss)`, `Legea binomială negativă (nbin)`,`Legea geometrică (geo)`.

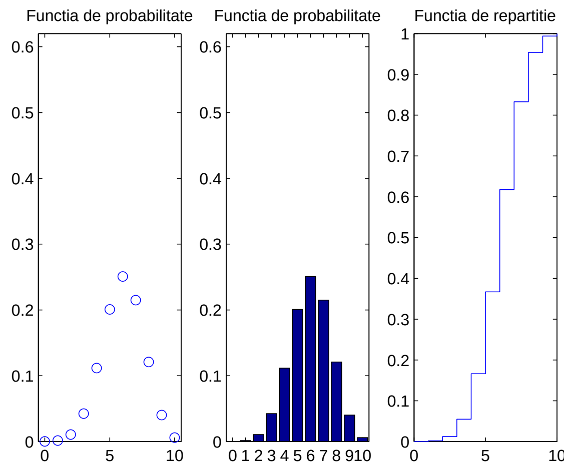
Exemplu

Să se scrie un program Matlab care să reprezinte grafic funcţia de probabilitate (prin puncte şi bare) şi funcţia de repartiţie ale legii de probabilitate binomială (***bino***).

Execuţia programului pentru `n=10 p=0.6`:

```
n=input('n=');
p=input('p=');
x=0:n;
f=pdf('bino',x,n,p);
subplot(1,3,1),plot(x,f,'o')
axis([-0.5 n+0.5 0 max(p)+0.02])
title('Functia de probabilitate')
subplot(1,3,2),bar(x,f)
axis([-0.5 n+0.5 0 max(p)+0.02])
title('Functia de probabilitate')
f=cdf('bino',x,n,p);
subplot(1,3,3),stairs(x,f)
title('Functia de repartitie')
axis([0 n 0 1])
```

realizează graficele următoare:



9.2. Legi de probabilitate continue

Definiţie Fie variabila aleatoare X având funcţia de repartiţie F. Vom spune că X este variabilă aleatoare absolut continuă, dacă funcţia de repartiţie F este absolut, sau echivalent se poate reprezenta sub forma:

$$F(x)=\int_{-\infty}^{x} f(t)\,dt,$$

Pentru $\forall\ x\in R$, funcţia $f: R\rightarrow R$, numindu-se **densitatea de probabilitate** a variabilei aleatoare X.

Definiţie Fie vectorul aleator X având funcţia de repartiţie F. Spunem că X este **vector aleator de tip continuu**, dacă funcţia de repartiţie F se poate reprezenta sub forma:

$$F(x)=F(x_1,x_2, \ldots x_n)=\int_{-\infty}^{x1} \ldots \int_{-\infty}^{xn} f(t1, t2, \ldots tn)\, dt1 \ldots dtn$$

$\forall\ x\in R^n$, funcţia f: $R^n\rightarrow R$, numindu-se densitatea de probabilitate a vectorului aleator X.
Forma cea mai generală a densităţii de probabilitate a unei variabile aleatoare din clasa respectivă, ne dă o lege de probabilitate de tip continuu.
Funcţiile **pdf, cdf** prezentate în paragraful anterior se utilizează şi în acest caz.
Următoarele legi de probabilitate continue clasice pentru care sistemul Matlab are implementate funcţii sunt:
Legea uniformă (unif), Legea normală (norm), Legea lognormală (logn), Legea gamma (gam), Legea exponenţială (exp), Legea beta, Legea Weibull (weib) Legea Rayleight (rayl), Legea t (Student) necentrată.

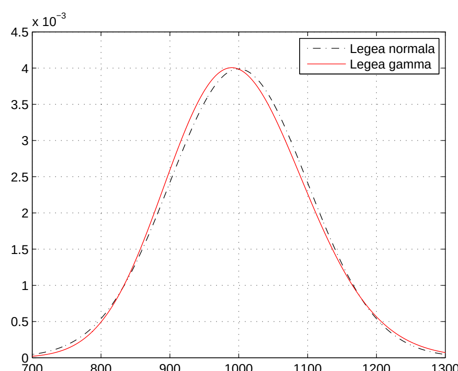
Exemplu

Să se reprezinte pe acelaşi grafic, densităţile de probabilitate pentru legile normală şi gamma.
Executând programul pentru a=100 şi b=10 obţinem:

Codul Matlab

```
clf; a=input('a=');b=input('b=');
m=a*b;s=b*sqrt(a);
x=m-3*s:0.01:m+3*s;
fn=pdf('norm',x,m,s);
fg=pdf('gam',x,a,b);
plot(x,fn,'k-.',x,fg,'r-')
grid
legend('Legea normala','Legea gamma')
```

rezultă graficul:



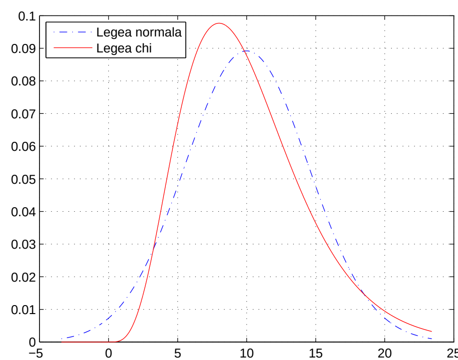
9.3. Legi de probabilitate continue statistice

Legile de probabilitate continue statistice denumite astfel în sistemul Matlab prin pachetul de programe *Statistics toolbox,* deoarece utilizarea este strâns legată de statistică.
Matlab-ul are implementate funcţii pentru următoarele legi de probabilitate:
Legea `t (Student)` (t), Legea Hemert-Pearson `(chi2)`, Legea `F(Fisher-Snedecor)(f)`, Legea F(Fisher-Snedecor) necentrată (ncf).

Exemplu
Să se scrie un program Matlab care să reprezinte pe aceaşi figură graficele densităţilor de probabilitate pentru **legile normală şi chi2.**
Prin execuţia programului Matlab pentru n=10:

```
clf;
n=input('n='); s=sqrt(2*n);
x=n-3*s:0.01:n+3*s;
fn=normpdf(x,n,s);fchi=chi2pdf(x,n);
plot(x,fn,'b-.',x,fchi,'r-')
grid
legend('Legea normala','Legea chi',2);
```

se obţine graficul:

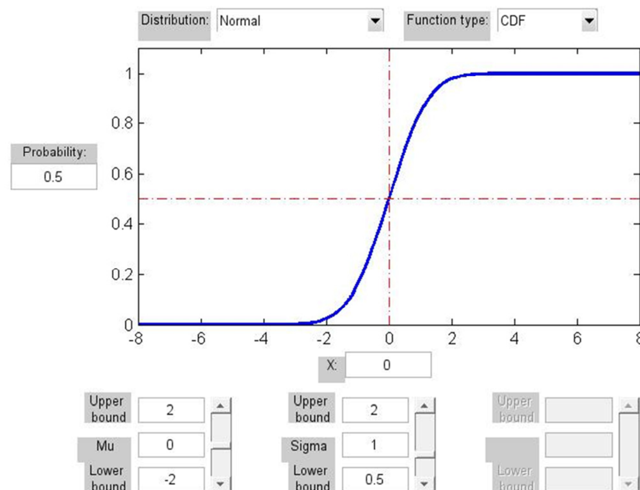


9.4. Funcţia Matlab `disttool`

Funcţia (comanda) `disttool` este un program demonstrativ. Lansarea în execuţie a acestuia se face din linia de comandă :

```
>> disttool
```

În urma căreia se produce o fereastră grafică interactivă (demonstrativă) privind funcţia de repartiţie `cdf` şi funcţiile de probabilitate `pdf` .



9.4.1. Funcţia Matlab `normspec`

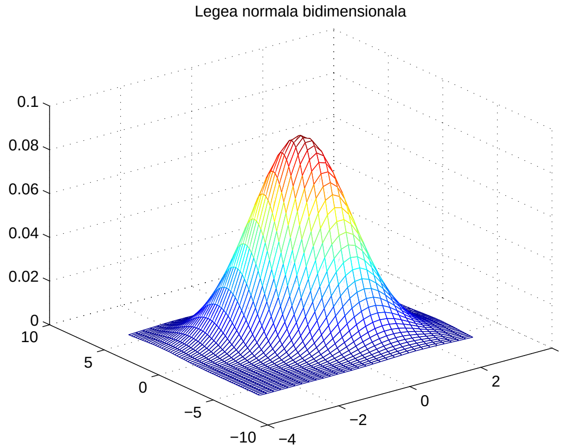
Funcţia **normspec** reprezintă grafic densitatea de probabilitate a legii normale de parametrii mu şi sigma şi umbreşte aria mărginită de dreptele perpendiculare pe axa absciselor ce trec prin punctele axei precizate prin cele două componente ale vectorului sp, curba ce reprezintă grsficul densităţii de probabilitate şi axa absciselor.
Parametrul p, după execuţia funcţiei, conţine valoarea ariei umbrite, adică probabilitatea ca variabila aleatore să ia valori în intervalul precizat prin sp.
Moduri de apel:

```
Normspec(sp,mu,sigma)
P=normspec(sp,mu,sigma)
```

Exemplu de program Matlab care reprezintă grafic densitatea de probabilitate a legii normale bidimensionale pentru `m1=m2=0,s1=1,s2=2 şi r=-0.5.`

```
%date de intrare
clf,clear
m1=input('mu1=');% in workspace se introduce 0
m2=input('mu2=');% in workspace se introduce 0
s1=input('sigma1=');
s2=input('sigma2=');
r=input('r(-1<r<1):');
x=m1-3*s1:.2:m1+3*s1;
y=m2-3*s2:.2:m2+3*s2;
%trasare grafic
[X,Y]=meshgrid(x,y);
Z=1/(2*pi*s1*s2*sqrt(1-r^2))*exp(-1/(2*(1-r^2))...
    *((X-m1).^2/s1^2-2*r*(X-m1).*(Y-m2)/(s1*s2)...
    +(Y-m2).^2/s2^2));
mesh(X,Y,Z)
title('Legea normala bidimensionala');
```

Se obţine graficul:



9.5. Caracteristici numerice

Definiţie Numim *valoare medie* sau *speranţă matematică caracteristica numerică*

$$E(x) = \int_{-\infty}^{\infty} x dF(x)$$

Unde integrala Stieltjes se impune să fie absolut convergentă pentru ca valoarea medie să existe.

Definiţie Numim *dispersie sau varianţă caracteristica numerică* ataşată variabilei aleatoare X definită prin:

$$Var(x) = E[(X-E(X))^2]$$

Definiţie Dacă se consideră vectorul aleator bidimensional (X,Y) numim *covarianţă sau corelaţie* caracteristica numerică :

$$Cov(X,Y)=E[(X-E(X))(Y-E(Y))]$$

Respectiv *coeficient de corelaţie*, raportul :

$$r(X,Y) = \frac{Cov(X,Y)}{\sqrt{Var(X)Var(Y)}}$$

Sistemul Matlab dispune de proceduri pentru calcul valorilor medii şi ale dispersiilor legilor de probabilitate implementate prin *Statistics toolbox [15].*

Se apelează astfel:

```
[m,v]=numef(x,par1,par2,...)
```

Unde `numef` este un şir de caractere, care definesc legea de probabilitate din care ultimele patru sunt `stat,` iar cele ce le preced sunt cele care dau numele legii.

În urma executării instrucţiunii, se calculează matricile m şi v ale valorilor medii şi ale dispersiilor pentru legea considerată, având parametrii precizaţi prin matricile `par1,par2,...`

Aceste matrice trebuie să fie de aceleaşi dimensiuni, cu excepţia că dacă sunt scalari, aceştia se extind la matricele constante şi care iau valorile scalarilor corespunzători.

Definiţie Numim *funcţie caracteristică* ataşată variabilei aleatoare X, respectiv vectorului aleator X, funcţia definită prin:

$$\varphi(t)=M(e^{itX}), \quad \forall\ t\in R$$

Utilitatea funcţiei caracteristice este dată de netezimea ei, care este superioară funcţiei de repartiţie, acesta din urmă putând fi discontinuă, pe când funcţia caracteristică este uniform continuă. În plus, pe baza formulei de inversiune, există o caracterizare completă a distribuţiei unei variabile aleatoare sau a unui vector aleator cu ajutorul funcţiei caracteristice.

Definiţie Fie variabila aleatoare X . Numim *moment iniţial* respectiv *moment centrat* de ordin n, caracteristicile numerice:

$$\nu_n = E(X^n), \mu_n = E[(X-E(X)^n] = E[(X-\nu_1)^n].$$

Definiţie Fie variabila aleatoare X, care are funcţia de repartiţie F. Numim *mediană* caracteristica numerică `m`, care
satisface condiţiile:

$$P(X \geq m) \leq \frac{1}{2} \leq P(X \leq m)$$

Având în vedere proprietăţile condiţiile de mai sus se pot scrie echivalent:

$$F(m-0) \leq \frac{1}{2} \leq F(m)$$

Definiţie Fie variabila aleatoare X, care are funcţia de repartiţie F. Numim *cuantilă de ordin* $\gamma \in (0,1)$ caracteristica numerică x_γ care satisface condiţiile :

$$P(X \geq x_\gamma) \geq 1-\gamma, \quad P(X \leq m) \geq \gamma$$

Ca şi în cazul `medianei` aceste condiţii se pot scrie echivalent sub forma :

$$F(x_\gamma - 0) \leq \frac{1}{2} \leq F(x_\gamma)$$

Funcţia Matlab `icdf`

Pentru calculul cuantilelor este necesară inversarea funcţiei de **repartiţie.** Sistemul Matlab prin *Statistics toolbox* dispune de funcţii pentru inversarea funcţiei de repartiţie ale legilor de probabilitate implementate.

Apelarea acestor funcţii se poate face cu una din formele:

$$x=icdf('legea', P, p_1, p_2, \ldots)$$
$$x=numef(P, p_1, p_2, \ldots)$$

unde `legea` este un şir de caractere predefinit pentru fiecare din legile de probabilitate disponibile în *Statistics toolbox,* `numef` este un şir de caractere din care ultimele trei sunt `inv`, iar cele ce le preced sunt cele care dau numele predefinit al legii de probabilitate (ca şi parametrul `legea`) .

În urma executării uneia din cele două instrucţiuni, se calculează matricea `x` a cuantilelor legii `precizată` prin parametrii `legea, respectiv numef,` corespunzătoare valorilor date prin matricea `P` şi având parametrii daţi prin matricele `par1,par2,...` Aceste matrice trebuie să fie de aceleaşi dimensiuni, cu excepţia că, dacă sunt scalari, caz în care aceştia se extind la matricele constante şi care iau valorile scalarilor corespunzători.

Exemplu

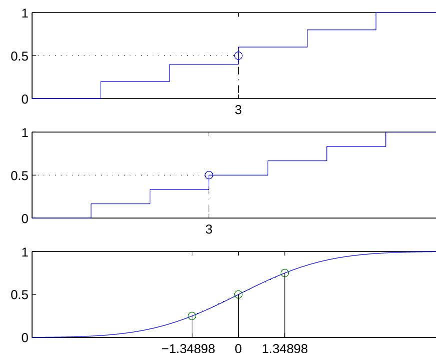
Să se scrie un program Matlab care calculează **mediana** pentru legea uniformă discretă şi reprezintă acest lucru pentru două valori distincte ale parametrului `N`, precum şi cuartilele legii normale.

Executarea programului Matlab următor pentru `N=5` şi `N=6`, respectiv `mu=0,sigma=2,`

```
clf,clear
N1=input('N1=');N2=input('N2=');
m=input('mu=');s=input('sigma=');
xu1=0:N1+1;yu1=unidcdf(xu1,N1);
xu2=0:N2+1;yu2=unidcdf(xu2,N2);
xn=m-3*s:0.01:m+3*s;yn=normcdf(xn,m,s);
me1=icdf('unid',1/2,N1);
me2=icdf('unid',1/2,N2);
Q=icdf('norm',[1/4,2/4,3/4],m,s);
subplot(3,1,1),stairs(xu1,yu1);
set(gca,'Xlim',[0,N1+1]),set(gca,'xtick',me1)
set(gca,'xticklabel',me1),hold on
plot([0,me1],[1/2,1/2],'k:',me1,1/2,'o')
```

```
plot([me1,me1],[0,1/2],'k-.')
subplot(3,1,2),stairs(xu2,yu2);
set(gca,'Xlim',[0,N2+1]),set(gca,'xtick',me2)
set(gca,'xticklabel',me2),hold on
plot([0,me2],[1/2,1/2],'k:',me2,1/2,'o')
plot([me2,me2],[0,1/2],'k-.')
%%%%
subplot(3,1,3),
plot(xn,yn,[Q(1),Q(2),Q(3)],[1/4,2/4,3/4],'o')
set(gca,'Xtick',[Q(1),Q(2),Q(3)])
set(gca,'xticklabel',[Q(1),Q(2),Q(3)]),hold on
X=[m-3*s,Q(1);m-3*s,Q(2);m-3*s,Q(3)];
plot(X,[1/4,1/4;1/2,1/2;3/4,3/4],'k:')
plot([Q(1),Q(1)],[0,1/4],'k-')
plot([Q(2),Q(2)],[0,2/4],'k-')
plot([Q(3),Q(3)],[0,3/4],'k-')
```

are ca rezultat :



9.6. Statistică Matematică

Statistica matematică se ocupă cu descrierea şi analiza numerică a fenomenelor de masă, dezvăluind particularităţile lor de volum, structură, dinamică, conexiune, precum şi regularităţile sau legile care le guvernează. Recomand lucrarea de referinţă [28].

Etapele principale ce se disting în cercetarea statistică a unui fenomen aleator se pot considera ca fiind:

a) *Definirea (concepţia)* obiectului studiat, care conţine definirea unităţilor statistice, conceperea chestionarului (întrebărilor), planificarea culegerii datelor.
b) *Observarea fenomenului (culegerea datelor)* conform criteriilor stabilite la etapa precedentă.
c) *Descrierea statistică,* ce cuprinde reprezentarea grafică a datelor statistice, sistematizarea acestora, precum şi calcularea indicatorilor numerici pentru punerea în evidenţă a unor proprietăţi şi pentru sugerarea unor ipoteze referitoare la legile care guvernează fenomenul cercetat.
d) *Modelarea probabilistică* a fenomenului cercetat, care are ca obiectov principal cercetarea fenomenului folosind ca instrument de lucru teoria probabilităţilor.

9.6.1. Concepte de bază ale statisticii

Definiţie Numim *colectivitate (populaţie)* o mulţime C de elemente cercetată din punct de vedere auneia sau mai multor proprietăţi, elementele componente numindu-se *indivizi sau unităţi statistice.*
Definiţie Numim *caracteristică sau variabilă* a colectivităţii C proprietatea supusă investigării statistice relativă la C.
O caracteristică X ce ia o mulţime cel mult numărabilă de valori o numim caracteristică de tip *discret,* iar dacă ia valori dintr-un interval finit sau infinit o numim caracteristică de tip *continuu.*
Modurile de culegere şi colectare a datelor statistice utilizate sunt:

a) *Observarea totală (exhaustivă, recensământ)* când toţi indivizii colectivităţii C sunt înregistraţi.
b) *Observarea parţială (sondaj,selecţie),* când după criterii bine stabilite sunt înregistraţi o parte din indivizii colectivităţii C, numim *eşantion sau selecţie.*
c) *Observarea curentă*, când înregistrarea indivizilor se efectuează odată cu apariţia (producerea lor).
d) *Observarea periodică,* când înregistrarea fenomenului se efectuează la intervale de timp stabilite.

9.6.2. Funcţii Matlab

Funcţia `random`

Instrucţiunile prin care sunt generate numere aleatoare ce urmează una din legile de probabilitate recunoscute de sistemul MATLAB® prin *Statistic toolbox* sunt de forma:

```
X=random('legea',par1,par2,…,m,n)
X=numef(par1,par2,...,m,n)
```

unde `'legea '` este un şir de caractere predefinit pentru fiecare din legile de probabilitate disponibile prin *Statistic toolbox, nume f* este un şir de caractere din care ultimele trei sunt `rnd,` iar cele care le preced sunt cele care dau numele predefinit al legii de probabilitate.
Executarea uneia din cele două forme ale instrucţiunii, are ca efect generarea matricei X , cu m linii şi n coloane, de numere aleatoare ce urmează legea de probabilitate corespunzătoare cu parametrii precizaţi prin: `par1,par2,...`
Aceste matrici trebuie să fie de aceleaşi dimensiuni cu matricea X, cu excepţia că dacă unele din aceştia sunt scalari, caz în care aceştia se extind la matricele constante de aceleaşi dimensiuni cu celelate şi care iau valorile scalarilor corespunzători. Dacă toţi parametrii sunt scalari, când lipsesc parametrii `m` şi `n` se generează un număr aleator, când lipseşte `n,` atunci trebuie ca `m` să fie un vector cu două componente, care conţine dimensiunile matricei X.

Funcţiile mvnrd şi mvtrnd

Funcţiile `mvnrd şi mvtrnd` sunt singurele funcţii din *Statistic toolbox* prin care se generează legi de probabilitate multidimensionale, respectiv legile normală şi legea Student (t). Apelurile se fac:

```
X=mvnrnd(mu,v,m)
X=mvtrnd(r,n,m)
```

În primul caz, sunt generaţi m vectori aleatori ce urmează legea normală multidimensională, având vectorul valorilor medii `mu` şi matricea covarianţelor `v (matrice pozitiv definită).`

Cei `m` vectori aleatori generaţi sunt obţinuţi în matricea X care va vea `m` linii, iar numărul coloanelor este dat de dimensiunea legii de probabilitate şi care trebuie să coincidă cu lungimea vectorului `mu` şi cu ordinul matricei pătrate v.

În al doilea caz vor fi generaţi `m` vectori aleatori ce urmează legea ***Student*** multidimensională, parametrul `r` reprezintă matricea coeficienţilor de corelaţie, care este matrice pozitiv definită având pe diagonala principală toate elementele 1, iar parametrul `n` reprezintă numărul gradelor de libertate, putând fi un scalar sau un vector de lungime `m.`Trebuie remarcat faptul că o linie a matricei X se obţine dintr-un vector aleator ce urmează legea normală multidimensională, având valoarea medie 0 şi matricea covarianţelor dată prin parametrul `r,` ale cărui componente se împart la un număr aleator ce urmează legea χ^2.

Funcţiile rand , randn, randperm

Sistemul Matlab de bază conţine două funcţii pentru generarea de numere aleatoare ce urmează legea uniformă `U(0,1)` prin `rand,` respectiv legea normală standard `N(0,1),` prin `randn.`

Unele situaţii impun regăsirea unui anumit şir de numere aleatoare generat prin aceste două legi avem la dispoziţie ceea ce se numeşte `starea generatorului.`

Obţinerea stării generatorului la un moment dat se face astfel:

```
S=rand('state');
S=randn('state ');
```

Pentru reiniţializarea sau schimbarea stării generatorului se poat utiliza parametrul `sum(100*clock)` care este resetat la fiecare apel în funcţie de tipul calculatorului.

Generarea unei permutări a primelor `n` numere naturale se realizează prin instrucţiunea

```
Perm=randperm(n.
```

9.6.3. Tabele statistice

Definiţie Numim *tabel statistic (simplu, nesistematizat)* un tablou în care înregistrările sunt trecute în ordinea apariţiei lor.

Definiţie Numim *tabel statistic (sistematizat) relativ la caracteristica X de tip discret*, tabloul în care sunt trecute valorile distincte ale caracteristicii în care sunt trecute valorile distincte ale caracteristicii şi frecvenţele cu care au apărut aceste valori.

Definiţie Fie caracteristica de tip continuu X, care ia valori în intervalul (a, b) descompus în diferite intervale disjuncte, numite clase, prin punctele care satisfac relaţiile:

$$a=a_0 \le a_1 \le ... \le a_n = b$$

Numim ***tabel statistic (sistematizat)*** *relativ la caracteristica X,* tabloul ce conţine clasele caracteristicii şi frecvenţele cu care au apărut aceste clase.

Definiţie Numim *amplitudinea* clasei definită de intervalul $[a_i, a_{i-1}]$ lungimea acestui interval, iar frecvenţa relativă a clasei x_i , raportul $p_i = \frac{fi}{N}$

Definiţie Fie colectivitatea **C** relativ la care sunt cercetate două caracteristici X şi Y. Numim tabel de ***contingenţă,*** un tablou care conţine clasele caracteristicilor X şi respectiv Y, împreună cu frecvenţele absolute ale acestor clase.

9.6.4. Funcţii Matlab

Obţinerea de tabele sistematizate, în Matlab este posibilă cu ajutorul unor funcţii din *Statistic toolbox.*

Funcţiile tabulate, crosstab, caseread, casewrite, tblread, tblwrite
Prin apelul funcţiei:

`t = tabulate (x)`

se obţine în `t` (în lipsa acestui argument pe ecran) un tabel statistic sistematizat având trei coloane care conţine pe prima coloană conţine valorile distincte `x` , a doua frecvenţele absolute ale acestor valori distincte, iar ultima reprezintă frecvenţele relative în procente.
Prin executarea funcţiei:

`t = crosstab(x,y,z, ...)`

se obţine pe ecran respectiv în `t` un tabel statistic sistematizat cu atâtea intrări câţi vectori există. Dacă sunt doi parametrii `x,y` se obţine `tabelul de contingenţă.`
Pentru complectarea tabelului de contingenţă se utilizează funcţiile Matlab:
`caseread, casewrite,tblread, tblwrite.`
Apelul :

`casewrite(obs,'file')`

are ca efect **scrierea datelor** din matricea `obs` de tip caracter în fişierul cu numele `file.`

Citirea datelor se face prin apelul:

`obs=caseread('file')`

şi care ca efect citirea datelor din fişierul cu numele `'file'` în matricea `obs` de tip caracter.

Apelul :

`tblwrite(t,va,obs, 'file')`

are ca efect scrierea în fişierul cu numele `'file'` a matricelor `va,obs` de tip caracter şi a datelor din matricea numerică `t`.
Prin apelul:

`[t,va,obs]=tblread('file')`

Se va citi din fişierul `'file'` a matricelor `va,obs` de tip caracter şi a datelor din matricea numerică `t`.
`Exemplu [1,pag 126]`
Să se scrie un program Matlab care generează `N` vectori aleatori ce urmează legea normală bidimensională, după care să se construiască tabelul de corelaţie cu `m` clase în raport cu prima variabilă, respectiv `n` clase în raport cu a doua variabilă.
Codul Matlab este:

```
clear,mu(1)=input('m1=');mu(2)=input('m2=');
v(1,1)=input('sigma1^2=');
v(2,2)=input('sigma1^2=');
v(1,2)=input('Cov(X,Y)=');v(2,1)=v(1,2);
N=input('N=');m=input('m=');n=input('n=');
if det(v)<=0
    error('Matricea v nu este pozitiv definita ');
end
X=mvnrnd(mu,v,N);xx=X(:,1);yy=X(:,2);
xmin=min(xx);xmax=max(xx);
ymin=min(yy);ymax=max(yy);
cx=xmin:(xmax-xmin)/m:xmax;
cy=ymin:(ymax-ymin)/n:ymax;
```

```
for k=1:N
    for i=1:m-1
        if(cx(i)<=xx(k)) &(xx(k)<cx(i+1))
            x(k)=i;
        end
    end
    if (cx(m)<=xx(k))&(xx(k)<cx(m+1))
        x(k)=m;
    end
   for j=1:n-1
        if(cy(j)<=yy(k))&(yy(k)<cy(j+1))
            y(k)=j;
        end
    end
    if (cy(n)<=yy(k))&(yy(k)<cy(n+1))
        y(k)=n;
    end
end
t=crosstab(x,y);
disp(t);
```

Pentru: m1=5, m2=10, sigma1^2=2, sigma1^2=3, Cov(X,Y)=1, N=100, m=6, n=4 se obţine tabelul:

1	0	0	0	0
0	1	3	0	0
0	3	5	4	0
0	2	13	8	2
0	4	15	8	1
0	1	7	12	2
0	0	3	5	0

9.6.5. Reprezentări grafice

Vom da în continuare câteva din cele mai importante definiţii, necesare acestui paragraf care se pot regăsii de exemplu în **[1, pag: 129-228].**

Definiţie: Numim *diagramă prin batoane (bare) a unei distribuţii statistice* X de tip discret, reprezentarea grafică într-un sistem de axe rectangulare a segmentelor (batoanelor) date prin $\{(x_i, y)/\ 0 \leq y \leq \alpha f_i\}$, i=1,…n, $\alpha > 0$ fiind un factor de proporţionalitate, iar f_i este frecvenţa absolută a valorii x_i.

Definiţie Numim *diagramă cumulativă (ascendentă)* a unei distribuţii statistice X de tip discret, linia poligonală care uneşte punctele de coordonate $(x_i, \alpha F_i)$, unde F_i este frecvenţa cumulată (ascendentă) ataşată valorii x_i , iar $\alpha > 0$ fiind un factor de proporţionalitate.

Definiţie Numim *histograma* unei distribuţii statistice X de tip continuu, diagrama obţinută prin construirea de dreptunghiuri având drept baze clasele distribuţiei statistice şi înălţimile astfel considerate încât ariile dreptunghiurilor şă fie proporţionale cu frecvenţele claselor.

Dacă factorul de proporţionalitate este 1/N , atunci se obţine histograma frecvenţelor relative. Vom nota cu f_s funcţia în scară ce delimitează aceste dreptunghiuri.

Definiţie Numim *poligonul frecvenţelor* al unei distribuţii statistice X de tip continuu, poligonul obţinut prin unirea punctelor de coordonate $(x_i, \alpha_i f_i)$, i=1,…n, unde α_i fiind un factor de proporţionalitate, iar f_i este frecvenţa clasei x_i.

Definiţie Numim *diagrame integrale (cumulative)* ale frecvenţelor cumulate ascendente, respectiv descendente, relative la distribuţia statistică de tip continuu liniile poligonale obţinute prin unirea punctelor de coordonate (α_k ,F_k) ,respective: (α_k ,F'_k), k=1,2,…n.

Definiţie Numim *nor statistic* ataşat caracteristicilor X şi Y, punctele din plan obţinute prin reprezentarea grafică a datelor primare şi este utilizat pentru observarea formei legăturii funcţionale care există între cele două caracteristici.

Funcţii Matlab pentru reprezentarea grafică a datelor statistice
Reprezentările grafice pentru distribuţia statistică a unei caracteristici (variabile) de tip continuu se obţin prin utilizarea funcţiilor `plot,bar,hist,histc`.

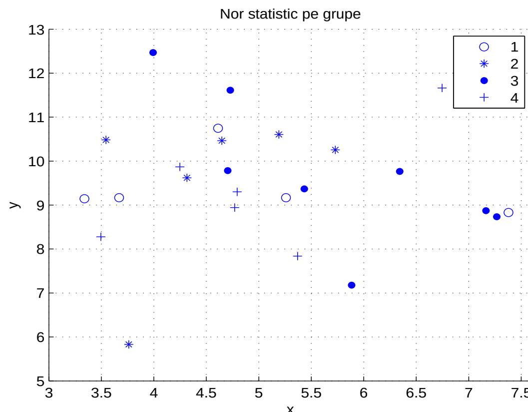
Funcţiile scatter, scatter3, gscatter
Primele două realizează norului statistic în cazul bidimensional (respectiv tridimensional), iar a treia este disponibilă în *Statistic toolbox,* dar în plus reprezintă norul statistic în cazul bidimensional pe grupe de puncte.

Exemplu
Să se scrie un program Matlab care generează n vectori aleatori ce urmează legea normală bidimensională şi n numere aleatoare ce urmează legea uniformă discretă.
Folosind aceste date să se reprezinte norul vectorilor aleatori bidimensionali, efectuând gruparea conform numerelor aleatoare uniforme generate.

Codul Matlab

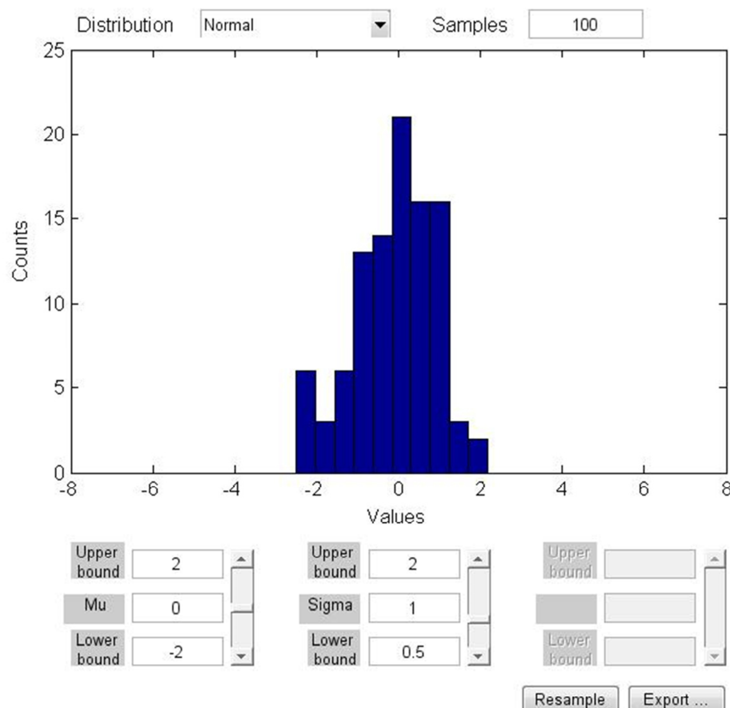
```
clear,mu(1)=input('m1=');mu(2)=input('m2=');
v(1,1)=input('sigma1^2=');
v(2,2)=input('sigma1^2=');
v(1,2)=input('Cov(X,Y)=');v(2,1)=v(1,2);
if det(v)<=0
    error('Matricea v nu este pozitiv definita ');
end
N=input('N(parametrul legii uniforme)=');
n=input('n=');X=mvnrnd(mu,v,n);
x=X(:,1);y=X(:,2);
g=unidrnd(N,n,1);
gscatter(x,y,g,'b','o*.+')
grid
title('Nor statistic pe grupe')
```

Pentru valorile de intrare : `m1=5, m2=10, sigma1^2=2, sigma1^2=3,Cov(X,Y)=-1, N(parametrul legii uniforme)=4,n=25` se obţine graficul:



9.6.6. Funcţia randtool

Aceasta este o funcţie demonstrativă pusă la dispoziţie de Matlab care după ce se lansează în execuţie prin comanda:
`>> randtool`
produce o fereastră grafică intercativă (demonstrativă) privind generarea numerelor aleatoare şi ilustrarea acestora cu histograme. Fixarea legii de probabilitate în scop demonstrativ se face prin alegerea din meniul legilor de probabilitate situat în partea stângă sus a a ferestrei, iar volumul numerelor aleatoare, ce urmează a fi generate, se precizează prin introducerea acestuia în fereastra din partea dreaptă sus. Limitele pentru parametrii legii de probabilitate considerate pot fi precizate prin introducerea acestora în ferestrele considerate.
Activarea butonului `output` are ca effect salvarea numerelor aleatoare `ans` sau în variabila precizată de utilizator, iar butonul `resample` permite repetarea generării de numere aleatoare cu acelaşi volum şi aceaşi parametric.



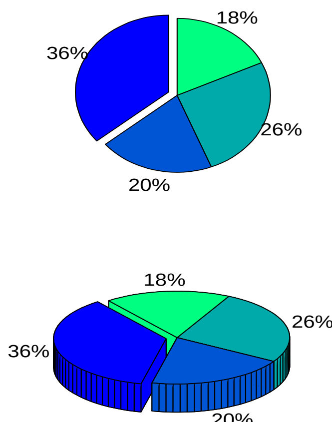
Funcţiile pie, pie3

Aceste funcţii realizează reprezentarea datelor prin sectoare circulare în plan, respective în spaţiu.
Exemplu
Să se scrie un program Matlab care generează n numere aleatoare ce urmează legea uniformă discretă, construieşte tabelul sistematizat, după care reprezintă datele sistematizate folosind **funcţiile pie, pie3.**
Codul Matlab

```
n=input('n=');N=input('N=');
x=unidrnd(N,1,n);
```

```
t=tabulate(x);
d=length(t(:,3));
ex=zeros(1,d);
ex(1)=1;
subplot(2,1,1), pie(t(:,3)/100,ex)
subplot(2,1,2), pie3(t(:,3)/100,ex)
colormap winter
```

Pentru `n=50, N=4` obţinem graficul:



9.6.7. Parametrii distribuţiilor statistice

Parametrii statici ce măsoară tendinţa

Definiţie

Media *aritmetică* a distribuţiei statistice a caracteristicii X este dată prin:

$$X_n^m = \sum_{k=1}^{n} p_k\, x_k$$

Definiţie

Media *geometrică* a distribuţiei statistice a caracteristicii X este dată prin:

$$\log X_n^m = \sum_{k=1}^{n} p_k \log x_k$$

Definiţie

Media *armonică* a distribuţiei statistice a caracteristicii X este dată prin:

$$X_n^h = \frac{1}{\sum_{k=1}^{n} p_k \frac{1}{x_k}}$$

Între cele trei medii are loc ingalitatea :

`Media armonică ≤ Media geometrică ≤ Media aritmetică`

Funcţiile mean, geomean , harmean, trimmean, median

Matlabul conţine funcţia `mean,` iar `Statistic toolbox` cele două funcţii `geomean şi hartmean` şi au ca efect calculul mediilor corespunzătoare pentru un vector dat x, iar dacă x este o matrice are ca efect calculul mediilor pentru fiecare coloană a matricei.

Funcţia `trimean` are ca efect calculul mediei aritmetice a vectorului x, după ce au fost eliminate cele mai mici p% componente, respectiv p% cele mai mari elemente, iar dacă x este o matrice are ca efect calculul mediilor pentru fiecare coloană a matricei.
Apelul este:

```
M=trimean(x,p)
```

Definiţie
Numim *mediana* distribuţiei statistice a caracteristicii X, valoarea numerică m care împarte datele statistice ordonate crescător în două părţi.
Funcţia `median` ale sistemului de bază Matlab are ca efect calculul medianei corespunzătoare pentru un vector dat x, iar dacă x este o matrice are ca efect calculul mediilor pentru fiecare coloană a matricei.

Parametrii statistici ce măsoară dispersia
Statistic toobox conţine funcţia `prctile` care calculează pentru componentele vectorului x, după ce acesta a fost ordonat crescător, centilele precizate prin parametru p care conţine unul sau mai multe numere întregi de la 1 la 99.
Definiţie
Numim *moment de ordin k* v_k a distribuţiei statistice a caracteristicii X, valoarea numerică :

$$V_k=\sum_{i=1}^{n} p_i \ x_i^k$$

Definiţie
Numim *amplitudine* (interval de variaţie) a a distribuţiei statistice a caracteristicii X, valoarea numerică:

$$\varpi = x_{max} - x_{min}$$

Iar *abatere medie (absolută)* a distribuţiei statistice a caracteristicii X valoarea numerică:

$$\delta = \sum_{k=1}^{n} p_k \mid x_k - x' \mid$$

unde: $x' = x_a$.
Definiţie
Numim *moment centrat de ordin k* v_k a distribuţiei statistice a caracteristicii X, valoarea numerică :

$$\mu_k = \sum_{i=1}^{n} (xi - x)^k$$

Funcţia moment
Apelul funcţiei `moment` se poate face prin:

```
S = moment (x,k)
```

Şi are ca efect calculul, pentru un vector dat x a momentului centrat de ordin k, iar dacă x este o matrice are ca efect calculul mediilor pentru fiecare coloană a matricei.

Funcţia grpstats
Funcţia `grpstats` din `Statistic toolbox` are ca efect determinarea unor parametri statistici pentru date clasificate după o anumită variabilă de grupare.
Apelul se face astfel:

$$[m_a, s, fr, nume] = grpstats(x, g)$$

unde g este un vector numeric sau de tip caracter, de aceaşi lungime cu numărul liniilor matricei x şi care permite gruparea datelor. În plus de mai calculează abaterile standard s, numărul datelor `fr` şi etichetele `nume` pentru fiecare grupă.

Exemplu [1, pag163]
Să se scrie un program Matlab care generează 100 de numere aleatoare ce urmează legea uniformă discretă , cu N=4. În matricea x cu 100 linii şi 3 coloane, vor fi generate numere

aleatoare ce urmează legea normală, unde parametrul σ =1, iar parametrul μ este respectiv 1,2 şi 3 pentru cele trei coloane.
Apoi folosind vectorul de grupare `g` se calculează valorile medii, abaterile standard, frecvenţele absolute şi să se precizeze numele grupelor, pentru coloanele matricei x.

Codul Matlab

```
g=unidrnd(4,100,1);
t=1:3;
t=t(ones(100,1),:);
x=normrnd(t,1);
[ma,s,fr,nume]=grpstats(x,g);
fprintf(' Mediile pe grupe \n')
disp(ma)
fprintf(' Abaterile standard pe grupe \n')
disp(s)
fprintf(' Frecventele pe grupe \n')
disp(fr)
fprintf(' Etichetele pe grupe \n')
disp(nume)
```

În urma execuţiei programului se obţin următoarele rezultate:

```
Mediile pe grupe
   1.0161   1.9818   3.1427
   0.9819   1.9379   3.0399
   1.0187   2.1575   3.0173
   1.4505   2.6751   2.9265

Abaterile standard pe grupe
   0.1442   0.1588   0.1948
   0.1712   0.2311   0.1766
   0.2323   0.2369   0.1876
   0.3279   0.1940   0.3269

 Frecventele pe grupe
   34   34   34
   26   26   26
   24   24   24
   16   16   16

 Etichetele pe grupe
   '1'
   '2'
   '3'
   '4'
```

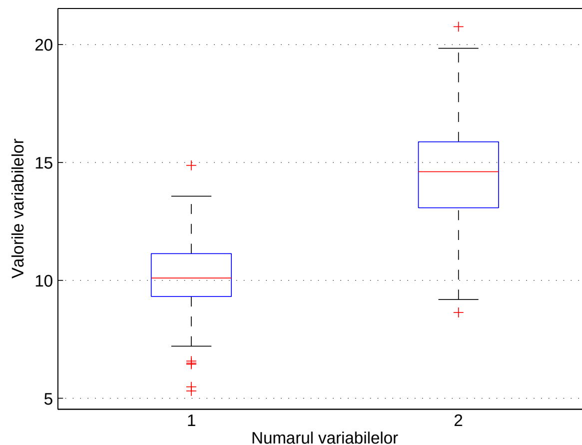
Funcţia boxplot

Sistemul Matlab, prin *Statistics toolbox* , dispune de funcţia `boxplot`, efectul executării ecstei instrucţiuni este reprezentarea grafică prin dreptughiuri cu prelungiri de segmente pentru fiecare coloană a matricei x.

Exemplu
Să se scrie un program Matlab care generează n vectori aleatori care urmează legea normală bi-dimensională, în matricea `x` de tipul `nX2`, după care folosind funcţia `boxplot`, reprezintă datele pentru prima variabilă şi respectiv pentru cea de a doua variabilă.

Codul Matlab
```
clear,mu(1)=input('m1=');mu(2)=input('m2=');
v(1,1)=input('sigma1^2=');
v(2,2)=input('sigma2^2=');
v(1,2)=input('Cov(X,Y)=');v(2,1)=v(1,2);
if det(v)<=0
    error('Matricea v nu este pozitiv definita ');
end
n=input('n=');
X=mvnrnd(mu,v,n);
boxplot(X);
grid
xlabel('Numarul variabilelor'),
ylabel('Valorile variabilelor')
```

În urma executării programului cu valorile:
m1=10, m2=15, sigma1^2=4, sigma2^2=5, Cov(X,Y)=-3, n=100 se obţine graficul :



9.6.8. *Corelaţie şi regresie*

Corelaţia poate fi înţeleasă ca fiind legătura care există între o caracteristică dependentă şi una sau mai multe caracteristici independente, iar *regresia* este metoda prin care se stabileşte această legătură.
În acest paragraf ne vom limita prin a prezenta câteva definiţii esenţiale, iar pentru detalii suplimentare se poate consulta [28].

Definiţie
Numim *moment* de ordin (k_1,k_2) *al distribuţiei statistice a caracteristicii bidimensionale* (X,Y) valoarea numerică :
$\overline{v}_{k1,k2} = \sum_{i=1}^{m}\sum_{j=1}^{n} p_{ij}\, x_i^{k1} y_j^{k2}$, unde $p_{ij} = \frac{f_{ij}}{N}$ este frecvenţa relativă a clasei (x_i , y_j)
Definiţie
Numim *moment* centrat de ordin (k_1,k_2) al distribuţiei statistice bidimensionale (X,Y), *valoarea numerică* :

$$\bar{\mu}_{k1,k2} = \sum_{i=1}^{m}\sum_{j=1}^{n} p_{ij} \quad (x_i-\bar{x})^{k1}(y_i-\bar{y})^{k2} \text{ unde}$$
$$\bar{x} = \overline{v}_{10} = \frac{1}{N}\sum_{i=1}^{m} f_i \; x_i$$
$$\bar{y} = \overline{v}_{01} = \frac{1}{N}\sum_{j=1}^{n} f_i \; y_i$$

Definiţie
Numim *coeficient de corelaţie* **Pearson** al distribuţiei statistice bidimensionale (X,Y) , raportul:

$$r = \frac{\bar{\mu}_{11}}{\sqrt{\mu_{20}\mu_{02}}}$$

Definiţie
Numim *valoare medie condiţionată* a distribuţiei statistice a caracteristicii Y în raport cu X = x_j, valoarea numerică:
$\bar{y}_i = \bar{y}(x_i)$, respectiv *valoarea medie condiţionată a distribuţie statistice* a caracteristicii X în raport cu Y= y_j, valoarea numerică $\bar{x}_j = \bar{x}(y_j)$.

Definiţie
Curba de ecuaţie y = f(x) pe care se situează punctele de coordonate (x_i, $\bar{y}_i$) , i=1,2,...m se numeşte *curba de regresie* a lui Y în raport cu X, iar *curba de ecuaţie* x = g(y) pe care se situează punctele de coordonate ($\bar{x}_i$,y_j), j= 1,...,n se numeşte *curba de regresie* a lui X în raport cu Y.
Pentru determinarea *curbelor de regresie* se poate folosii de exemplu ***metoda celor mai mici pătrate*** [1, pag 172].

Observaţie
Tipuri de **curbe de regresie care pot fi liniarizate** :

- $y = ab^x$ care prin logaritmare se liniarizează astfel: log y=log a+xlog b, lând z=log y, A=log a, B=log b.
- y = a/x+b (hiperbolică)
- $y = \frac{1}{\frac{a}{x}+b}$, care se liniarizează cu $u = \frac{1}{x}$, $v = \frac{1}{y}$;
- y= a log x+b (logaritmică), care se liniarizează cu z=log x;
- $y=be^{ax}$ (exponenţială), care se liniarizează lny= ln b+ ax, luând z= lny;
- $y=bx^a$ care se liniarizează log y=logb+a logx luând u=logx, v=logy;
- $\frac{1}{y} = a\, e^{-x}+b$, (caz particular al *legăturii logistice)* care se liniarizează dacă se ia

$u = e^{-x}$, $v = \frac{1}{y}$

Există **funcţii ce nu pot fi liniarizate**:

- $y = ax^b + c\log x$,
- $y = ax^b + e^{cx}$
- $y = a + bx + ce^{dx}$

Definiţie
Centrul de greutate al distribuţiei statistice (X,Y) este punctul de intersecţie a două drepte de regresie.

Funcţiile `cov, corrcoef,isline,refline, gline, nlifit`
Funcţia `cov` calculează matricea covarianţelor pentru matricea X, unde fiecare coloană este considerată a fi o variabilă.
Instrucţiunea `diag(cov(X))` generează vectorul varianţelor fiecărei coloane a matricei X, iar `sqrt(diag(cov(X)))` este vectorul abaterilor standard pentru fiecare coloană a matricei X.
Funcţia `corrcoef` calculează matricea coeficienţilor de corelaţie pentru matricea X.

Exemplu
Să se genereze `n` vectori aleatori, care urmează *legea normală bidimensională,* în matricea X de tipul `nX2`, după care folosind funcţiile **`cov` şi `corrcoef`**, calculează matricele covarianţelor şi a coeficienţilor de corelaţie, pentru cele două coloane ale matricei X.

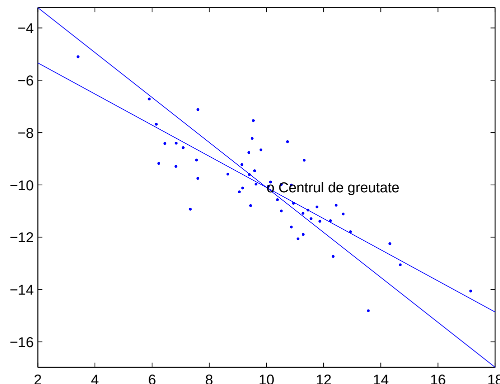
Codul Matlab:
```
clear,mu(1)=input('m1=');mu(2)=input('m2=');
v(1,1)=input('sigma1^2=');
v(2,2)=input('sigma1^2=');
v(1,2)=input('Cov(X,Y)=');v(2,1)=v(1,2);
n=input('n=');
if det(v)<=0
    error('Matricea v nu este pozitiv definita ');
end
X=mvnrnd(mu,v,n);
v=cov(X);
r=corrcoef(X);
fprintf('Matricea covariantelor \n'),disp(v)
fprintf('Matricea coeficientilor de corelatie \n'),disp(r)
```
Pentru `mu1=100,mu2=-100, v=[5 4; 4 4], n=1000` se obţin rezultatele:
```
Matricea covariantelor
    4.7652    3.8133
    3.8133    3.8659

Matricea coeficientilor de corelatie
    1.0000    0.8885
    0.8885    1.0000
```

Funcţia *isline* reprezintă grafic în figura curentă dreptele de regresie, obţinute cu metoda celor mai mici pătrate pentru fiecare curbă a figurii , care a fost reprezentată cu instrucţiunea *plot,* dar nu prin linie continuă sau de tipul linie continuă.
Funcţia *refline* reprezintă grafic în figura curentă dreapta de ecuaţie `y = mx+n`, dacă lipsesc argumentele efectul funcţiei coincide cu *isline.*
Funcţia *gline* reprezintă grafic în figura precizată prin parametrul f, iar dacă acesta lipseşte în figura curentă, segmentul de dreaptă prin marcarea cu ajutorul mouse-ului a capetelor segmentului.
Funcţia *nlifit* pusă la dispoziţie de *Statistics toolbox* pentru ajustarea neliniară, bazată pe metoda celor mai mici pătrate.

Exemplu
Să se scrie un program Matlab, care generează N vectori aleatori ce urmează *legea normală bidimensională,* reprezintă norul statistic pentru aceste date, precum şi cele două drepte de regresie, a lui Y în raport cu X , respectiv a lui X în raport cu Y.

Codul Matlab:
```
clear,mu(1)=input('m1=');mu(2)=input('m2=');
v(1,1)=input('sigma1^2=');
v(2,2)=input('sigma1^2=');
v(1,2)=input('Cov(X,Y)=');v(2,1)=v(1,2);
N=input('N=');
if det(v)<=0
    error('Matricea v nu este pozitiv definita ');
end
Z=mvnrnd(mu,v,N);
X=Z(:,1);Y=Z(:,2);
ma=mean(Z);
s=std(Z);
r=corrcoef([X,Y]);
m=s(2)/(r(1,2)*s(1));
n=ma(2)-m*ma(1);
plot(X,Y,'.'),lsline,refline(m,n)
text(ma(1),ma(2),'o Centrul de greutate')
```
Cu datele de intrare:
`m1=10, m2=-10, sigma1^2=8, sigma1^2=3, Cov(X,Y)=-4, N=50`
Se obţine graficul:



9.6.9. Funcţia interactivă `nlintool`

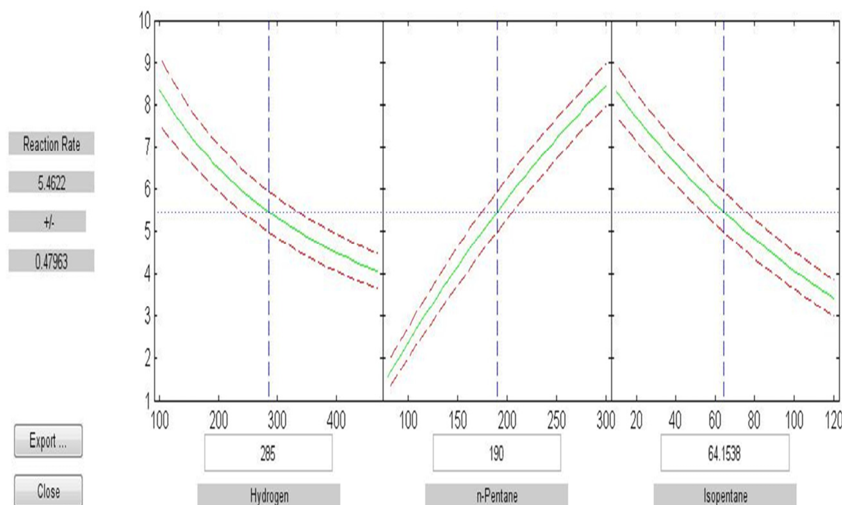
Se lansează în linia de comandă prin:
`>> nlintool(x,y,numef,a0);`
În urma lansării se produce un grafic interactiv privind ajustarea datelor conţinute în vectorul coloană y şi matricea x, având tipul legăturii precizat prin **nume f**, care poate fi o funcţie Matlab proprie.
Pe figură este reprezentat graficul de ajustare cu linie continuă. Prin introducerea valorii unui argument x în fereastra de jos sau prin deplasarea dreptei verticale pe poziţia abscisei x, pe axa ordonatelor se obţine valoarea corespunzătoare pentru y.

Elementele obţinute, prin lansarea acestei funcţii, pot fi salvate în spaţiul de lucru, `workspace`, parţial sau în totalitate, prin utilizarea butonului `export`.
Pe lângă aceste elemente, graficul demonstrativ conţine şi alte elemente printre care *intervalele de încredere.*

Exemplu preluat din Help
Dacă tastăm în linia de comandă:

```
>>load reaction
>>nlintool(reactants,rate,@hougen,beta,0.01,xn,yn)
```

Se obţine :



9.7. Teoria Selecţiei

9.7.1 Tipuri de selecţie

Definiţie
Numim *eşantion (selecţie, sondaj)* relativ la colectivitatea C o submulţime de indivizi ε ∈ C, care urmează să fie cercetaţi din punct de vedere a uneia sau mai multor caracteristici, iar numărul indivizilor din eşantionul ε se numeşte *volumul eşantionului.*
Clasificare
După modul de obţinere :

1. metode nealeatoare
2. metode aleatoare

de selecţie.
Metode nealeatoare:

1. *Selecţia sistematică* , când indivizii care intră în eşantion sunt consideraţi după o anumită regulă, de exemplu din 10 în 10.

2. *Selecţie tipică,* când, cunoscându-se informaţii anterioare la colectivitate, sunt consideraţi indivizi cu valori medii apropiate de valoarea medie a întregii colectivităţi.
3. *Selecţie stratificată,* când colectivitatea este stratificată (clasificată) după anumite criterii, cunoscându-se proporţia indivizilor pentru fiecarea strat. Eşantionul se ia astfel încât să fie respectate aceste proporţii pentru fiecare strat.

Metodele aleatoare de selecţie = fiecare individ al colectivităţii C poate să intre în eşantion cu aceaşi probabilitate sau cu probabilităţi diferite.

1. *Repetate (bernoulliene)* când individul ce intră în eşantion, după ce a fost cercetat, este reintrodus în colectivitate.
2. *Nerepetate* când individul care intră în eşantion, după ce a fost cercetat, nu este reintrodus în colectivitate

Observaţie

Dacă volumul colectivităţii este mult mai mare decât volumul eşantionului, atunci o selecţie nerepetată poate fi considerată ca fiind de tip repetat.

Aceasta are la bază rezultatul privind comportarea asimptotică a legii *hipergeometrice* ca şi o lege binomială.

9.7.2. Funcţii de selecţie

Vom considera în cele ce urmează că avem de fiecare dată o selecţie repetată.

Definiţie

Fie colectivitatea C cercetată din punct de vedere al caracteristicii X. Numim *date de selecţie* relative la X datele statistice $x_1,x_2, ...,x_n$ privind indivizii care intră în eşantion.

Definiţie

Numim *funcţie de selecţie* sau *statistică* variabila aleatoare

$$Z_n=h_n\ (X_1,X_2,\ ...,\ X_n)$$

Unde $h_n:\Re^n\rightarrow\Re$ este o funcţie măsurabilă, iar $z_n=h_n\ (x_1,x_2, ...,x_n)$ se numeşte valoarea funcţiei de selecţie.

Definiţie

Numim *medie de selecţie* funcţia de selecţie

$$\frac{1}{n}\sum_{k=1}^{n} X_k = \overline{X},\ \text{iar}\ \frac{1}{n}\sum_{k=1}^{n} x_k = x$$

Se numeşte *valoarea medie de selecţie.*

Demonstraţia acestor proprietăţi se poate găsii în lucrarea (1, pag 199 : 200).

Proprietate

Fie caracteristica X având valoarea medie m = E(X) şi dispersia σ^2=Var (X), atunci

$$E\ (X)=m,\ Var(X) = \frac{1}{n}\sigma^2$$

Proprietate

Fie caracteristica X având valoarea medie m = E(X) şi dispersia σ^2=Var (X), atunci statistica

$$Z_n = \frac{\overline{X}\ - m}{\sigma^{-\sqrt{n}}}$$

Converge în repartiţie la legea *normală* N (0,1), când $n\rightarrow\infty$, iar când X urmează legea normală N (m,σ) afirmaţia are loc pentru orice valoare a lui n.

9.7.3 Momente de selecţie

Definiţie
Numim ***moment de selecţie*** de ordin k funcţia de selecţie :

$$\frac{1}{n}\sum_{i=1}^{n} X_i^k = \nu_k \text{ , iar } \nu_k = \frac{1}{n}\sum_{i=1}^{n} x_i^k \text{ ,}$$

se numeşte **valoarea momentului de selecţie de ordin k**.

Definiţie
Numim ***moment centrat*** de selecţie de ordin k *funcţia de selecţie*:

$$\frac{1}{n}\sum_{i=1}^{n}(Xi - \overline{X})^k = \overline{\mu_k} \text{ , iar } \frac{1}{n}\sum_{i=1}^{n}(xi - \overline{x})^k = \overline{\mu_k}$$

se numeşte **valoarea momentului centrat de selecţie de ordin k.**

Definiţie
Numim ***dispersie de selecţie*** funcţia de selecţie:

$$\frac{1}{n-1}\sum_{k=1}^{n}(Xk - \overline{X})^2 = \overline{\sigma_k}^2 \text{ , iar valoarea numerică } \frac{1}{n-1}\sum_{k=1}^{n}(xk - \overline{x})^2 = \overline{\sigma_k}^2$$

Se numeşte ***valoarea dispersiei de selecţie***.

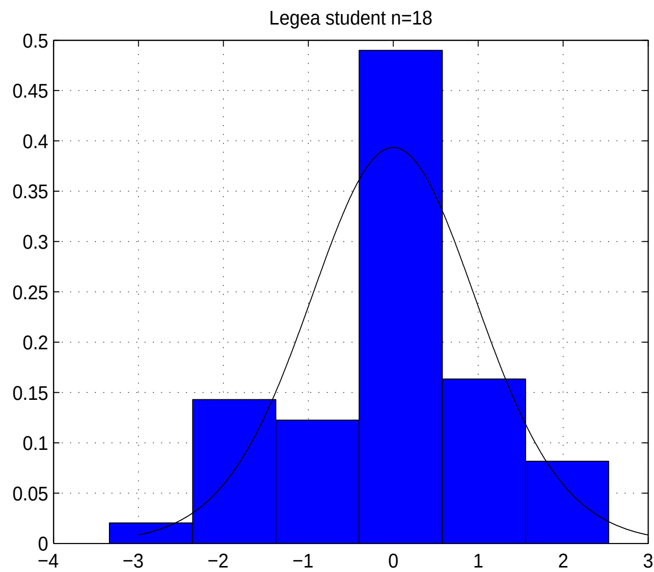
Exemplu
Să se scrie un program Matlab care generează de N ori câte n vectori aleatori, ce urmează legea normală bidimensională, având coeficientul de corelaţie dintre cele două componente nul, adică r=0. Folosind aceste date se vor calcula N numere aleatoare după regula precizată prin statistica :

$$T = \sqrt{n-2}\,\frac{\bar{r}}{\sqrt{1-\bar{r}^2}}$$

iar histograma corespunzătoare acestor noi date va fi reprezentată grafic împreună cu densitatea de probabilitate a legii *Student* cu n-2 grade de libertate.

Codul Matlab
```
clear,clf,
mu(1)=input('m1=');mu(2)=input('m2=');
v(1,1)=input('sigma1^2=');
v(2,2)=input('sigma1^2=');
v(1,2)=input('Cov(X,Y)=');v(2,1)=v(1,2);
if det(v)<=0
    error('Matricea v nu este pozitiv definita ');
end
N=input('N=');n=input('n=');
for k=1:N
    Z=mvnrnd(mu,v,n); r=corrcoef(Z);r12=r(1,2);
    t(k)=sqrt(n-2)*r12/sqrt(1-r12^2);
end
x=-3:0.01:3; f=tpdf(x,n-2);
nn=fix(1+10/3*log10(N));
[fr,c1]=hist(t,nn); h=c1(2)-c1(1);
bar(c1,fr/(h*N),1),hold on, plot(x,f, 'k-')
grid
colormap winter
```

Pentru : `m1=5, m2=10, sigma1^2=2, sigma1^2=3,Cov(X,Y)=0,N=50,n=20` rezultă graficul:



CAPITOLUL 10

REZOLVAREA ECUAŢIILOR DIFERENŢIALE ORDINARE

10.1. Metode numerice pentru problema Cauchy

Considerăm problema cu valori iniţiale (problema Cauchy).
Să se determine o funcţie cu valori vectoriale $\psi \in X^{1}[\alpha\,\beta]$, $\psi : [\alpha\,\beta] \rightarrow R^{d}$, astfel încât

$$\frac{dy}{dt} = f(t, y), t \in [a\ b]$$

$$y\,(a\,) = y_0$$

Vom evidenţia două clase importante de astfel de probleme:

1) pentru $\delta = 1$ avem o singură ecuţie diferenţială scalară de ordinul I

$$\psi' = f(t, y)$$

$$\psi(\alpha) = \psi_{0'}$$

2) pentru $\delta > 1$ avem un sistem de ecuaţii diferenţiale ordinare de ordinul I

$$\frac{dy^i}{dt} = f^i(t, y^1, y^2, \ldots y^d), \qquad i = 1, \ldots d$$

$$y^i\,(a) = y_0^{\,i}$$

Reamintim următoarea teoremă clasică referitoare la existenţă şi unicitate.

Teorema [4]

Presupunem că $\phi(\xi\,\psi)$ *este continuă în prima variabilă pentru* $\tau \in [\alpha\,\beta]$ *şi în raport cu cea de-a doua variabilă satisface o condiţie Lipschitz uniformă:*

$$kf(t, y) - f(t, y^*)k \leq Lky - y^*k \ \forall\, y, y_* \in R^d,$$

unde $k \cdot k$ *este o anumită normă vectorială. Atunci problema Cauchy are o souţie unică ,* $\psi(\tau), \alpha \leq \tau \leq \beta\ \forall\ \psi_0 \in R^d$. *Mai mult,* $\psi(\tau)$ *depinde continuu de* α *şi* ψ_0.

Se face distincţie între *metode de aproximare analitice* şi *metode discrete*.
În cadrul primei categorii se încearcă să se găsească aproximaţii $\psi_\alpha(\tau) \approx \psi(\tau)$ ale soluţiei exacte, valabile pentru orice $\tau \in [\alpha\,\beta]$. Acestea de obicei au forma unei dezvoltări într-o serie trunchiată, fie după puterile lui τ, fie în polinoame ortogonale Cebâşev, Laguerre etc. fie într-un alt sistem de funcţii de bază de exemplu B-splines. În cazul metodelor discrete, se încearcă să se găsească aproximaţii $\upsilon_\nu \in R^d$ ale lui $\psi(\tau_\nu)$ pe o grilă de puncte $\tau_\nu \in [\alpha\,\beta]$.
Abscisele τ_ν pot fi predeterminate (de exemplu puncte echidistante pe $[\alpha\,\beta]$), sau mai convenabil sunt generate dinamic ca parte a procesului de integrare.

Cea mai simplă metodă numerică , cunoscută ca metoda lui ***Euler*** provine dintr-o dezvoltare în serie ***Taylor*** prin trunchiere după termenul liniar :

$$y(t+h) = y(t) + h\frac{dy(t)}{dt}$$

ceea ce conduce la formula de bază :

$$y_n = y_{n-1} + hf(t_{n-1}, y_{n-1})$$

unde $y_n \approx y(nh)$, $t_n = nh$, h este un pas ales.
Dacă derivata a doua a soluţiei este mărginită în modul de M, ***Dormand*** în lucrarea [5] arată că eroarea pe pas este de ordin h^2, iar eroarea totală pe intervalul (a,b) este limitată de $M*\frac{b-a}{2}*h$.
Metoda lui ***Euler*** ia drept valoare medie a pantei valoarea ei în t. O valoare mai bună este valoarea pantei în mijlocul intervalului $t+\frac{h}{2}$, adică

$$y(t+h) = y(t) + hf(t+\frac{h}{2}, y(t+\frac{h}{2}))$$

Această valoare în mijlocul intervalului se poate aproxima cu metoda lui Euler cu un pas astfel:

$$y(t+\frac{h}{2}) \approx y(t) + \frac{h}{2} f(t, y(t))$$

şi se obţine un algoritm (***Runge)*** de forma:

$$K_1 = f(t, y(t))$$
$$K_2 = f(t+\frac{h}{2}, y(t)+\frac{h}{2} K_1)$$
$$y(t+h) = y(t) + hK_2$$

iar eroarea pe pas este de ordinul lui h^3.
Metoda lui ***Heun*** de ordin 3

$$a := kf(t_n, y_n);$$
$$b := kf(t_n+k/3, y_n+a/3);$$
$$c := kf(t_n+2k/3, y_n+2b/3);$$
$$y_{n+1} = y_n + 1/4(a+3c);$$

Metodele generate de acest tip, numite metode ***Runge-Kutta*** constau dintr-o succesiune de stagii, fiecare evalând o valoare aproximativă a pantei soluţiei exacte. Pasul final avansează soluţia din t la t+h prin utilizarea unei unei sume ponderate de pantele calculate anterior.
Deci:

$$K_1 = f(t, y(t))$$
$$K_2 = f(t+c_2h, y(t)+ha_{2,1} K_1)$$
$$K_3 = f(t+c_3h, y(t)+ha_{3,1} K_1 + ha_{3,2} K_2)$$
$$...$$
$$K_s = f(t+c_sh, y(t)+ha_{s,1} K_1 + ha_{s,2} K_2 + ... + ha_{s,s-1} K_{s-1})$$
$$y(t+h) = y(t) + h(b_1 K_1 + b_2 K_2 + ... + b_s K_s)$$

unde s este numărul de stagii.
Aceste metode utilizează un pas fix h. Prin micşorarea lui creşte precizia, dar şi timpul de calcul. Se poate ca acest pas să fie micşorat numai acolo unde soluţia aproximativă variază rapid şi să-l menţinem mai mare în zonele cu o variaţie mai lentă, deci pasul h trebuie să poată fi modificat în urma calculelor şi în concordanţă cu comportarea soluţiei.
T. Petrila şi D.Trif în lucrarea [21, pag. 187[sugerează rularea a două metode diferite în paralel, una pentru propagarea soluţiei, iar cealaltă numai pentru a estima şi a controla eroarea.
Fehlberg a descoperit că există perechi de metode ***Runge-Kutta*** cu ordine diferite de trunchiere în care liniile principale din tabelele de tip ***Butcher*** ataşate coincid.
Prin aceasta, ajustarea pasului h se poate face utilizând numai o singură evaluare în plus a funcţiei *f*.

Pentru formulele de aproximare de tip ***Runge-Kutta,*** un aspect esenţial este că sunt ***metode cu un pas*** , adică soluţia aproximativă la un nivel ulterior de timp $t+h$ se calculează numai din soluţia la nivel current de timp t .
Vom da în continuare câteva noţiuni legate de stabilitatea metodelor cu un pas, pentru detalii suplimentare recomandăm lucrarările [4], [5].
O ***diviziune (grilă)*** a intervalului [a,b] este o mulţime de puncte $\{x_n\}_{n=0...N}$ care verifică:

$$a = x_0 < x_1 <\ldots <x_{N-1} <x_N = b$$

cu pasul :

$$h_n=x_{n+1}-x_n \quad , \quad n=0,1,\ldots,N-1.$$

Fineţea grilei este dată de relaţia :

$$| h |= \max h_n, n=0,1,2,...N-1.$$

Funcţia $u =\{u_n\}, u_n \in \Re^d$ definită pe grila de mai sus se numeşte ***funcţie grilă.*** O metodă cu un pas produce o funcţie grilă astfel ca $u \approx y$, unde $y =\{y_n\}$ este o funcţie grilă indusă de soluţia exactă a unei probleme Cauchy. Notăm cu Γ_h mulţimea grilelor care se pot defini pe intervalul [a,b].

Definiţie
O metodă cu un pas de forma:

$$x_{n+1}=x_n+h_n$$
$$u_{n+1}=u_n+h_n\Phi(x_n,u_n;h_n)$$

se numeşte **stabilă** pe intervalul [a,b], dacă există o constantă K pozitivă care nu depinde de h, astfel încât pentru orice două funcţii grilă v, w $\in \Gamma_h$ are loc:

$$\|v-w\| \leq K(\|v_0-w_0\|_\infty + \|R_h\, v - \mathrm{R_h\, w}\|_\infty)$$

$(R_h\, v)$ este operatorul residual:

$$(R_h\, v) \ := \frac{1}{h_n}(v_{n+1}-v_n) - \emptyset(x_n, v_n; h_n) \text{ şi}$$

se numeşte **convergentă** dacă:

$\|u-y\|_\infty \to 0$, dacă l h l$\to$ 0 unde y este soluţia exactă a problemei Cau chy.

L.N. Trefeten demonstrează că stabilitatea metodei ***Euler*** este mulţimea punctelor z, astfel încât $|1-z| > 1$, pentru ***regula trapezului*** este: $\left|\frac{2+z}{2-z}\right| < 1$, iar pentru ***metoda lui Heun*** este:

$$\left|1+z+\frac{z^2}{2}\right| < 1.$$

Pentru a înţelege mai bine rolul stabilităţii vom da un exemplu din (8, pag:15):

Exemplu
Soluţia următoarei probleme Cauchy:

$$y'=5(y-t^2)$$
$$y(0)=0.008$$

este :

$$y(t)=t^2+0.4t+0.008$$

Dar pentru o constantă arbitrară C soluţia generală este:

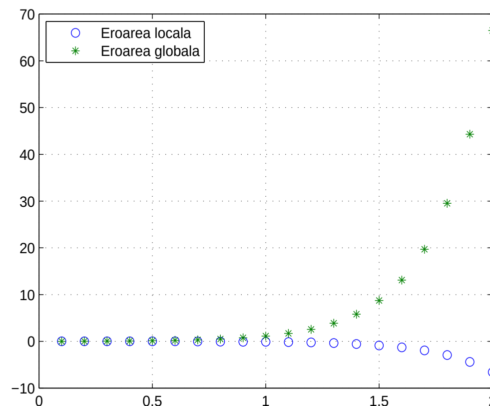
$$y(t)=(t^2+0.4t+0.008)+Ce^{5t}$$

Soluţia este instabilă deoarece pentru două constante arbitrare C_1, C_2 cele două soluţii diferă prin $(C_1-C_2)e^{5t}$. Această cantitate diferă foarte mult pentru valori mari ale lui t vizibilă în exemplul următor:

Codul Matlab

```
function Stab
% Diferenta dintre eroarea locală si cea globală
t = [];
le = [];
ge = [];
tn = 0;
yn = 0.08;
h = 0.1;
for i = 1:20
    tnp1 = tn + h;
    ynp1 = yn + h*ode(tn,yn);
    len = localsol(tn,yn,tn+h) - ynp1;
    gen = globalsol(tn+h) - ynp1;
    t = [t tnp1];
    le = [le len];
    ge = [ge gen];
    tn = tnp1;
    yn = ynp1;
end
plot(t,le,'o',t,ge,'*')
grid
axis([0 2 -10 70])
legend('Eroarea locala','Eroarea globala',2)
%================================================
function dydt = ode(t,y)
dydt = 5*(y - t^2);
function gy = globalsol(t)
% solutia globala.
gy = t^2 + 0.4*t + 0.08;
function ly = localsol(tn,yn,t)
%  solutia locala.
ly = globalsol(t) + (yn - globalsol(tn))*exp(5*(t - tn));
```

Rezulta graficul:



După efectuarea câtorva paşi cu asemenea metode, se poate utilize ***metode multipas***, [30] la care se utilizează informaţia de la mai multe nivele de timp anterioare.
Cele mai utilizate procedee sunt :

- ***Metoda Adams-Bashforth***

$$u_{n+1}=u_n+\frac{h}{12}\,(23f_n - 16f_{n-1} + 5f_{n-2}),\quad f_n= f(t_n,\ y_n)$$

care este o metodă explicită de ordin trei în raport cu pasul h.

- ***Metoda Adams-Moulton***

$$u_{n+1}=u_n+\frac{h}{24}\,(9f_{n+1} + 19f_n - 5f_{n-1} + f_{n-2})$$

este o metodă implicită, tot în trei paşi, dar de ordinul patru de exactitate în raport cu h.
Zona de stabilitate pentru ***Metodele Adams-Bashforth şi Adams-Moulton*** este:

$$|1 + z + z^2| < 1.$$

Vom scrie un program care stabileşte zona de stabilitate pentru ***Metoda Adams-Bashforth*** de ordin 3.

Cod Matlab

```
function adabash
c = [   1    0    0
      -3/2  3  -1/2 ];
d = [  23/12  -16/12  5/12
        3       0     0   ];
maxerror = zeros(10,2);
for m = 1:2
    for i = 2:10
        N = 2^i;
        h = 1/N;
        tn = 2*h;
         %y = [y_n, y_{n-1}, y_{n-2}]
        y = [exp(-2*h), exp(-h), exp(-0)];
        f = -y;
        for n = 3:N
            tnp1 = tn + h;
            ynp1 = dot(c(m,:),y) + h * dot(d(m,:),f);
            maxerror(i,m) = max(abs(ynp1 - exp(-
tnp1)),maxerror(i,m));
            y = [ynp1, y(1:2)];
            f = -y;
            tn = tnp1;
        end
    end
end
fprintf('\nFor h = 1/2^i, the maximum errors were\n\n')
fprintf('    i     AB3      Unstable\n')
fprintf('  --------------------------\n')
for i = 2:10
    fprintf('%5i %10.2e %10.2e\n',i,maxerror(i,:))
end
```

În urma executiei acestui program obtinem:

```
Pentru  h = 1/2^i, errorile maxime  sunt:
    i     AB3       Unstable
  --------------------------
    2   1.34e-003   9.68e-004
    3    2.31e-004    6.16e-003
    4    3.15e-005    1.27e+000
    5    4.08e-006    6.43e+005
    6    5.18e-007    2.27e+018
    7    6.53e-008    4.23e+044
    8    8.19e-009    2.27e+098
    9    1.03e-009    1.03e+207
   10    1.28e-010         Inf
```

Detalii asupra acestor metode precum şi asupra problemelor de convergenţă şi stabilitate se pot găsi în lucrarările [4], [5].

10.2. Rezolvarea numerică a problemelor cu valori iniţiale

MATLAB® -ul dispune de rezolvitori numerici care pot rezolva **problema Cauchy** atât pentru ecuaţii diferenţiale de ordin I (d=1) cât şi pentru sisteme de ecuaţii diferenţiale ordinare (d>1).
În tabelul următor sunt prezentate succint câţiva dintre aceşti rezolvitori [29, pp. 374:395]

Categorie	Funcţie	Descriere
Funcţii care rezolvă ODE	ode45	Rezolvă ecuaţii diferenţiale nonstiff, metodă de ordin mediu.
	ode23	Rezolvă ecuaţii diferenţiale nonstiff, metodă de ordin scăzut.
	ode113	Rezolvă ecuaţii diferenţiale nonstiff, metodă de ordin variabil.
	ode15s	Rezolvă ecuaţii diferenţiale stiff şi ecuaţii algebrice diferenţiale, metodă de ordin variabil.
	ode23s	Rezolvă ecuaţii diferenţiale stiff, metodă de ordin scăzut.
	ode23t	Ecuaţii diferenţiale stiff şi ecuaţii algebrice diferenţiale, metoda trapezelor.
	ode23tb	Rezolvă ecuaţii diferenţiale stiff, metodă de ordin scăzut.
Opţiuni ODE	odeset	Creează sau schimbă opţiuni de structură ale ODE.
	odeget	Permite obţinerea parametrilor din opţiunile ODE.
Funcţii de ieşire ODE	odeplot	Reprezentarea grafică a soluţiilor ODE (în funcţie de timp).

Aceşti rezolvitori se apelează cu una dintre sintaxele:
`[x,y] = ode xxx ('yprim', x0,xf,y0)`
`[x,y]= ode xxx ('yprim', x0,xf,y0, tol, trace)`
unde
xxx = poate fi 23, 45, 113, 15s, 23tb.
$[x_0,x_f]$ = intervalul de integrare
y_0 = este un vector coloană conţinând condiţiile iniţiale.
tol = este precizia dorită a soluţiei (opţională), implicit tol $=10^{-3}$ pentru *ode23* şi tol $= 10^{-6}$ pentru *ode45.*
trace = este un parametru care asigură tipărirea rezultatelor intermediare opţional.
Funcţiile *ode23, ode45* apelează la metode de integrare ***Runge-Kutta-Fehlberg*** cu stabilirea automată a pasului. Algoritmul ***Runge-Kutta*** cu pas automat îşi alege pasul funcţie de gradientul soluţiei, respectiv un pas mai mare, atunci când gradientul y'(x) are o valoare mai mică.

10.2.1. Probleme nonstiff

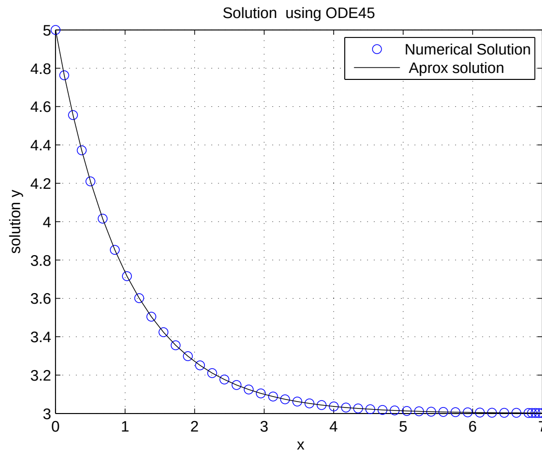
Definiţie
Se numeşte **soluţie neoscilantă** a unei probleme **Cauchy** o soluţie care are cel mult un zero în intervalul [a,b] şi **oscilantă** dacă are mai multe zerouri.
Observaţie
De interes sunt şi timpii de execuţie, de aceea alegerea rezolvitorului este eseneţială . Timpii de execuţie se pot obţine folosind funcţiile Matlab `tic-toc`.
Exemplu 2*:* (soluţie neoscilantă)
Considerăm ***problema Cauchy (P.V.I):***

$$dy/dt + y(t)=3, \quad y(0)=5.$$

cu soluţia exactă:

$$y(t)= 3+2\exp(-t).$$

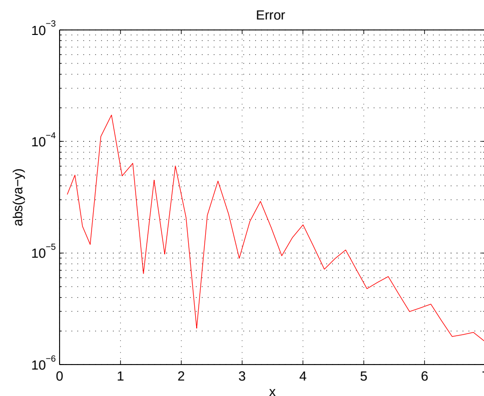
Aceasta modelează un proces de naştere-moarte pentru o populaţie de mărime y(t) .
Problema este preluată din lucrarea **[23, *pag. 171].***
Rata de naşteri este 3, iar variaţia populaţiei este proporţională cu populaţia. Procesul este dominat de decese (dy/dt=- y(t)+3) , iar populaţia iniţială 5 devine repede irelevantă.
Se crează un fişier funcţie:
```
function dy=f1(x,y)
dy=-y(t)+ 3
```
cu numele f1.m care se apelează cu **secvenţa MATLAB**:
```
[x,y]=ode 45( 'f1',[0,7], [0,5]);
ya=3+2* exp(-t)%soluţia analitică
plot (x, y, 'o', x, ya);
grid on
title('Solution using  ODE45');
xlabel(' x');
ylabel('solution y');
legend('Numerical Solution',' Aprox solution');
```
obţinându-se soluţia numerică comparativ cu cea analitică.



Dacă se doreşte evaluarea erorii :

```
semilogy(x,abs(ya-y),'-r')
xlabel('x','FontSize',10)
ylabel('abs(ya(x)-y)','FontSize',10)
grid on
title('Error ')
```

obţinându-se următorul rezultat:



Exemplul 3 (soluţie oscilantă)

Să se determine o soluţie numerică pentru următoarea problemă IVP **(problema pendulului) :**

$$\theta'' + \sin(\theta) = 0, \quad \theta \in [0, 10\pi]$$

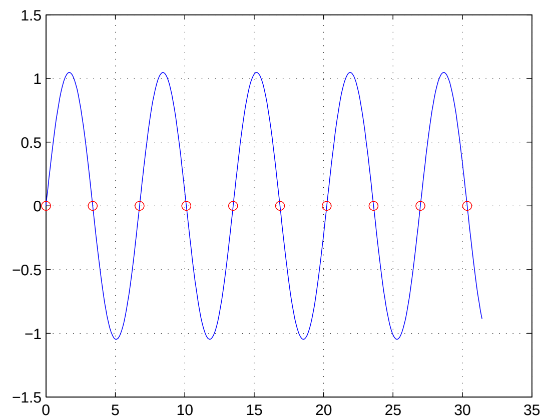
$$\theta(0) = 0, \quad \theta'(0) = 1$$

Deoarece soluţia oscilează de foarte multe ori în intervalul de integrare este obligatoriu de luat *RelTol=10*$^{-5}$, *şi AbsTol=10*$^{-10}$.

Codul Matlab

```
function pendul
tic;
options = odeset('Events',@event,'RelTol',1e-5,'AbsTol',1e-10);
[t,y,te,ye,ie] = ode45(@odes,[0 10*pi],[0; 1],options);
```

```
plot(t,y(:,1),te,ye(:,1),'ro')
grid on
spacing = diff(te)
toc;
%================================================
function dydt = odes(t,y)
dydt = [y(2); -sin(y(1))];
function [value,isterminal,direction] = event(t,y)
isterminal = 0;
direction = 0;
value = y(1);
```
Executând acest script se obţine graficul:



În cazul rezolvitorului `ode 45`, timpii de execuţie sunt: `0.190349 seconds.`
Dacă schimbăm rezolvitorul în `ode113`, vom obţine: `0.764025 seconds.`
Iar dacă rezolvitorul este `ode23`, atunci: `0.781178 seconds.`

Concluzie

Rezolvitorul `ode 45` are timpii cei mai buni, deci sugerăm folosirea acestui rezolvitor.

10.2.2. Probleme Stiff

La ecuaţiile diferenţiale ordinare de tip stiff (rigide) soluţiile pot avea variaţii foarte rapide în timp în raport cu intervalul de integrare şi este necesară folosirea unor paşi de integrare foarte mici, ceea ce nu este indicat la ecuaţiile nonstiff. Sugerăm folosirea rezolvitorului `ode15s`.

Pentru a ilustra modul de utilizare a acestui rezolvitori dăm în continuare următoarele exemple:

Exemplul 4.

O problema stiff cunoscută în literatura de specialitate este **modelul lui O'Maley :**

Fie problema Cauchy :

$$y' = y^2(1-y)$$
$$y(0)=\varepsilon,$$

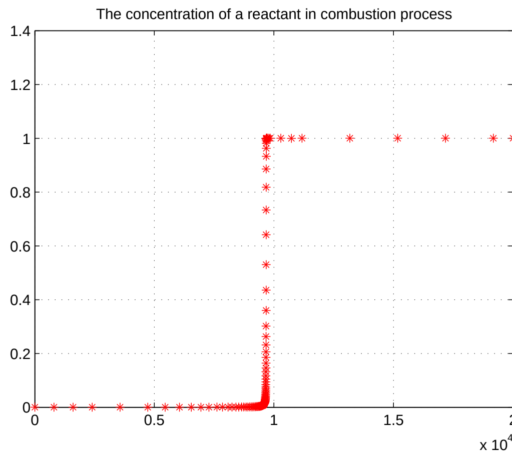
care trebuie rezolvată numeric pe intervalul $[0,\ 2/\varepsilon]$.

O'Maley fololoseşte **metoda perturbaţiei** pentru a analiza comportarea soluţiei pentru: $\varepsilon > 0$.

Codul MATLAB este:

```
function dydt = ode(t,y)
dydt=y^2*(1-y);
end
%%%%%%%%
epsilon =1e-4;
options = odeset('Stats','on');
[t,y]=ode15s(@ode,[0,2/epsilon],epsilon,options);
plot(t,y,'r*')
title('The concentration of a reactant in combustion process')
grid on
```

Rezolvând această problemă pentru $\varepsilon = 10^{-4}$ şi folosiind solverul *ode15s* bazat pe metode BDF (***backward differentiation formula***) obţinem următorul rezultat.



Exemplul 5

În lucrarea [5] ***Dormand J.*** propune un studiu comparativ al aplicării rezolvitorilor `ode45` şi `ode15s` pentru următorul sistem de ecuaţii diferenţiale cunoscut în literatura de specialitate ca **modelul lui Robertson.**

Acest model se mai numeşte şi modelul reacţiilor chimice consecutive şi este folosit ca model matematic în biologie şi medicină [23, pag. 154]:

$$\frac{\mathrm{d}y_1}{\mathrm{d}t} = -\alpha y_1(t) + \beta y_2(t) y_3(t),$$

$$\frac{\mathrm{d}y_2}{\mathrm{d}t} = -\alpha y_1(t) - \beta y_2(t) y_3(t) - \gamma y_2^2(t)$$

$$\frac{\mathrm{d}y_3}{\mathrm{d}t} = \gamma y_2^2(t)$$

Acesta modelează o reacţie chimică între trei reactanţi.

```
title('ode15s','FontSize',14)
```

Codul Matlab

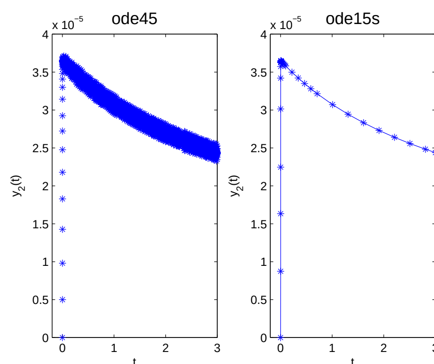
Câmpul vectorial

```
function yprime=chem(t,y,alpha,beta,gamma)
yprime=[-alpha*y(1)+beta*y(2)*y(3);
    alpha*y(1)-beta*y(2)*y(3)-gamma*y(2)^2;
    gamma*y(2)^2];
end
```

Scriptul Matalab

```
alpha=0.04;
beta=1e4;
gamma=3e7;
tspan=[0,3];y0=[1;0;0];
opts=odeset('Stats','on');
[ta,ya]=ode45(@chem,tspan,y0,opts,alpha,beta,gamma);
subplot(1,2,1),plot(ta,ya(:,2),'*')
ax=axis;ax(1)=-0.2;axis(ax);
xlabel('t'),ylabel('y_2(t)')
title('ode45','FontSize',14)
[tb,yb]=ode15s(@chem,tspan,y0,opts,alpha,beta,gamma);
subplot(1,2,2),plot(tb,yb(:,2),'-*')
axis(ax);
xlabel('t'),ylabel('y_2(t)')
```

În urma execuţiei programului rezultă graficul:




```
2052 successful steps
440 failed attempts
14953 function evaluations
33 successful steps
5 failed attempts
73 function evaluations
2 partial derivatives
13 LU decompositions
63 solutions of linear systems
>> disp([length(ta),length(tb)])
        8209          34
```

10.2.3. Singularităţi

Exemplul 5 [5 p. 130].
Davis (1962) foloseşte ecuaţia **lui** ***Emden:***

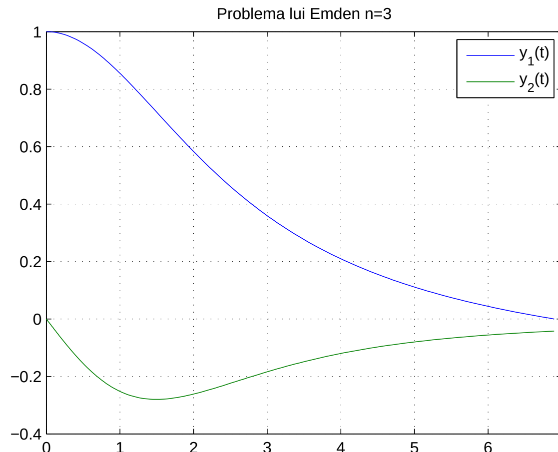
$$\frac{d^2y}{dx^2}+\frac{2}{x}\frac{dy}{dx}+y^n=0$$

pentru a modela comportarea termică a unui **nor de gaz**. O astfel de problemă are o singularitate evidentă în x=0, deci pentru a o putea rezolva decât dacă vom lua x = 0.1, iar pentru această valoare vom utiliza o aproximare analitică pentru $y(0.1)$, $y'(0.1)$.

Codul Matlab pentru n=3 este :

```
function Emden
% Problema lui Emden
global n
options = odeset('Events',@events,'RelTol',1e-8,'AbsTol',1e-10);
n = 3;
xend   = 10;
xinit  = 0;
yinit  = 1;
ypinit = 0;
[x,y,xe,ye,ie] = ode15s(@f,[xinit xend],[yinit; ypinit],options);
if isempty(ie)
    fprintf(' y(x) was never 0.')
else
    fprintf(' y(x)=0 for x = %g.\n',x(end))
end
plot(x,y(:,1),x,y(:,2))
title('Problema lui Emden n=3')
grid on
legend('y_1(t)','y_2(t)')
%==================================================================
function dydt = f(x,y)
global n
dydt = zeros(2,1);
dydt(1) = y(2);
dydt(2) = -1/3;
if x > 0
    dydt(2) = -(2/x)*y(2) - (y(1))^n;
end
function [value,isterminal,direction] = events(x,y)
value = y(1);
isterminal = 1;
direction  = 0;
```

Obţinem rezultatul:



Exemplul 6 [8, p. 131].
Fie ***modelul lui Kamke (1971) :***

$$y(y'')^2=e^{2x},\quad y(0)=0, y'(0)=0,$$

care descrie modificarea curenţilor într-un condensator cilindric.
Shampine , Gladwell şi Thompson în lucrarea [25] sugerează aproximarea soluţiei acestei probleme ***Cauchy*** folosind o dezvoltare în serie de forma:

$$y(x)= x^p(a+bx+...).$$

Se observă că această problemă are o singularitate în x = 0, deci ei sugerează că trebuie să completăm rezolvarea numerică a acesteia pentru valori mici ale lui x luând:

$$b = \frac{27}{40}a^{-2},$$

mai mult ei demonstrează că în această condiţie suplimentară există o singură soluţie de această formă.
Vom studia comparativ soluţiile obţinute folosind o dezvoltare în serie, apoi folosind rezolvitorul `ode45`.

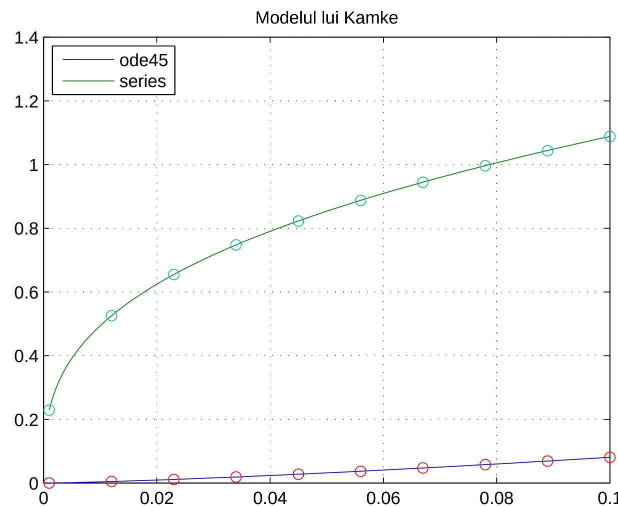
Codul Matlab

```
function kamke
x0 = 0.001;
[ys0,ysp0] = series(x0);
[x,y] = ode45(@odes,[x0 0.1],[ys0; ysp0]);
xs = linspace(x0,0.1,10);
[ys,ysp] = series(xs);
plot(x,y,xs,[ys; ysp],'o')
title('Modelul lui Kamke');
grid on
legend('ode45','series',2)
%================================================
function [ys,ysp] = series(x)
a = (81/16)^(1/3);
b = (27/40)/a^2;
ys = a*x.^(4/3) + b*x.^(7/3);
ysp = (4/3)*a*x.^(1/3) + (7/3)*b*x.^(4/3);
function dydx = odes(x,y)
dydx = [ y(2); sqrt(exp(2*x)/y(1)) ];
function dfdy = Jac1(x,y)
dfdy = [  0                                        1
         -1/2/sqrt(exp(2*x)/y(1))*exp(2*x)/y(1)^2  0 ];
```

```
function res = F2(x,y,yp)
res = [ yp(1) - y(2)
        yp(2) - exp(x)/sqrt(y(1)) ];

function [dfdy,dfdyp] = Jac2(x,y,yp)
dfdy = [ 0                      -1
         0.5*exp(x)*y(1)^(-3/2)   0 ];
dfdyp = [ 1    0
          0    1 ];
```

În acest exemplu am luat `x = 0.001` şi am dezvoltat în serii de puteri pentru a obţine o aproximare acceptabilă în `y(0.001)` şi `y′ (0.001)`. Am integrat în intervalul [0.001, 0.1] şi am obţinut rezultatul:



10.2.4. Opţiuni de structură a rezolvitorului

Cu ajutorul funcţiei `odeset` se poate stabili o structură de opţiuni ale rezolvitorului.
Forma generală este:

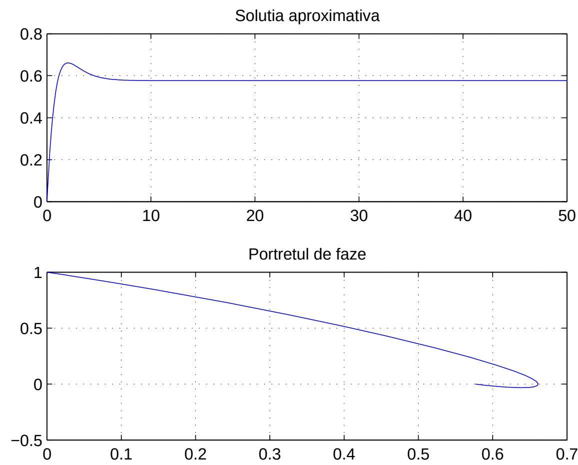
```
Options=odeset('name1',value1,'name2',value2,…)
```

Exemplul 7
Să se rezolve sistemul: y(1)'=y(2), y(2)'=-y(1)exp(-t)-y(2)
y(0)=0,y'(0)=1, si apoi sa se traseze portretul de faze, fară a folosii opţiunea odeset curba y' în funţie de y parametrizată de t.

```
%subprogram care descrie sistemul
function yp=fctd(t,y)
yp=zeros(2,1);
yp(1)=y(2);
yp(2)=-y(1)*exp(-t)-y(2);
```

```
%%%%%%%%%
%apelul functiei fctd
[t,y]=ode45(@fctd,[0,50],[0,1]);
subplot(2,1,1)
plot(t,y(:,1)); grid
title('Solutia aproximativa');
subplot(2,1,2)
plot(y(:,1),y(:,2));
title('Portretul de faze')
grid
```

În urma execuţiei programului rezultă graficul:



Exemplu 8

Să se reprezinte în planul fazelor modelul ***torului folsind opţiunea de rezolvare*** ***`odephase3`*** dat de următorul sistem de ecuaţii diferenţiale:

$$y_1' = -y_2 - \frac{y_3}{r} y_1$$

$$y_2' = y_1 - \frac{y_3}{r} y_2$$

$$y_3' = \frac{y_1}{r}$$

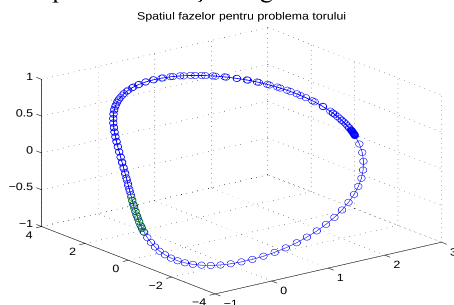
unde $r = \sqrt{y_1^2 + y_2^2}$

Condiţiile iniţiale sunt : $y_1(0) = 3$, $y_2(0) = 0$, $y_3(0) = 0$, $t \in [0, 10]$

Codul Matlab

```
function torus
options = odeset('OutputFcn',@odephas3,'Refine',10);
ode45(@f,[0 10],[3; 0; 0],options);
%================================================================
function dydt = f(t,y)
dydt = zeros(3,1);
r = sqrt(y(1)*y(1)+y(2)*y(2));
dydt(1) = -y(2) - (y(1) * y(3)) / r;
dydt(2) = y(1) - (y(2) * y(3)) / r;
dydt(3) = y(1) / r;
```

În urma execuţiei acestui script Matlab obţinem graficul:



Exemplu 9 [29, p. 378]

Fiind dat ***modelul lui Rosseller*** guvernat de următorul sistem de ecuaţii diferenţiale care apare în chimie şi modelează raportul dintre 3 reactanţi:

$$\frac{dy_1}{dt} = -y_2(t) - y_3(t)$$

$$\frac{dy_2}{dt} = y_1(t) + ay_2(t),$$

$$\frac{dy_3}{dt} = b + y_3(t)(y_1(t) - c)$$

a,b,c$\in \Re$.

Să se reprezinte în planul **fazelor (y_1(t), y_2(t))** şi **((y_1(t), y_2(t)),y_3(t)).**

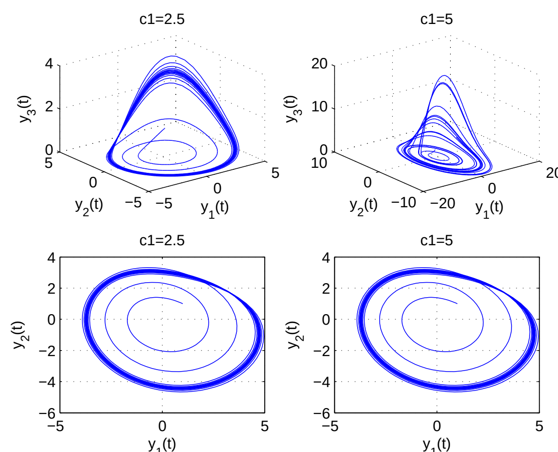
Câmpul vectorial este definit prin funcţia:

```
function yd = Roessler(t,y,a,b,c )
yd=[-y(2)-y(3);y(1)+a*y(2);b+y(3)*(y(1)-c)];
end
```

Sursa Matlab

```
tspan=[0,100];y0=[1;1;1];
options=odeset('AbsTol',1e-7,'RelTol',1e-4);
a=0.2;b=0.2;c1=2.5;c2=5;
[t,y]=ode45(@Roessler,tspan,y0,options,a,b,c1);
[t2,y2]=ode45(@Roessler,tspan,y0,options,a,b,c2);
subplot(2,2,1), plot3(y(:,1),y(:,2),y(:,3))
title('c1=2.5'), grid
xlabel('y_1(t)'),ylabel('y_2(t)'),zlabel('y_3(t)');
subplot(2,2,2), plot3(y2(:,1),y2(:,2),y2(:,3))
title('c1=5'), grid
xlabel('y_1(t)'),ylabel('y_2(t)'),zlabel('y_3(t)');
subplot(2,2,3), plot(y(:,1),y(:,2))
title('c1=2.5'), grid
xlabel('y_1(t)'),ylabel('y_2(t)');
subplot(2,2,4), plot(y(:,1),y(:,2))
title('c1=5'), grid
xlabel('y_1(t)'),ylabel('y_2(t)');
```

În urma execuţiei obţinem graficul:



10.2.5. Funcţiile deval, odextend

Dacă rezolvitorul se apelează cu comanda:

```
Sol=solver(odefun,tspan,z0,options)
```

Atunci acesta returnează o structură `sol` care poate fi evaluată cu ajutorul funcţiei `deval`.

Exemplul 10

Se dă problema Cauchy:

$$y' = y^2 - t, \quad t \in [0,3].$$

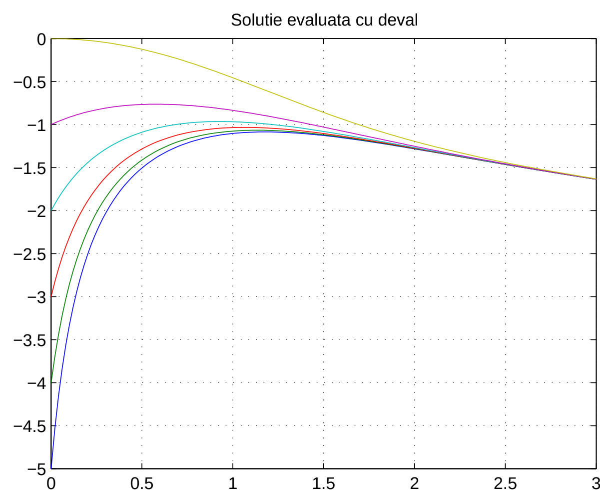
Cu valori iniţiale `y(0)=-5,-4,...,0.`

Codul Matlab

```
fd=@(t,y) y^2-t;
t=linspace(0,3,150);
y=zeros(150,6);
y0=-5:0;
for k=1:length(y0)
    sol=ode45(fd,[0,8],y0(k));
    y(:,k)=deval(sol,t,1);
end

plot(t,y)
title('Solutie evaluata cu deval')
grid
```

Se obţine graficul:

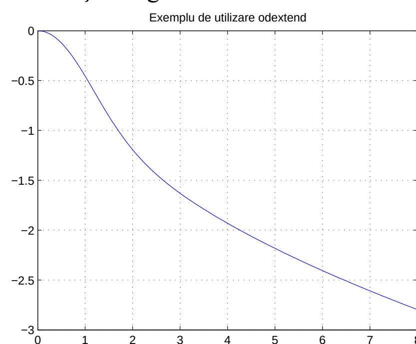


În acest exemplu **observăm că nu este necesar** să definim o funcţie pentru câmpul vectorial într-un fişier separat, ea poate fi inclusă în scriptul pe care-l lansăm în execuţie.

Funcţia *odextend* expandează soluţia unei probleme **Cauchy**. Pentru a exemplifica modul de lucru cu această funcţie vom scrie următorul program **Matlab.**

```
fd=@(t,y) y^2-t;
sol=ode45(fd,[0,3],0);
sol=odextend(sol,fd,8);
t=linspace(0,8,150);
y=deval(sol,t);
plot(t,y)
title('Exemplu de utilizare odextend')
grid
```

În urma execuţiei programului obţinem graficul:



10.2.6. Ecuaţii implicite

Rezolvitorul `ode15i` rezolvă probleme PVI implicite de forma:

$$F(t,y,y')=0$$

Utilizând o metodă BDF.
Sintaxa minimală este:

```
[t,y]=ode15i(odefun,tspan,y0,yp0)
```

Exemplu 11:
Să se rezolve problema lui ***Cauchy :***

$$y'^2+y^2-1=0$$

cu condiţia iniţială:

$$y(\pi/6)=1/2;$$

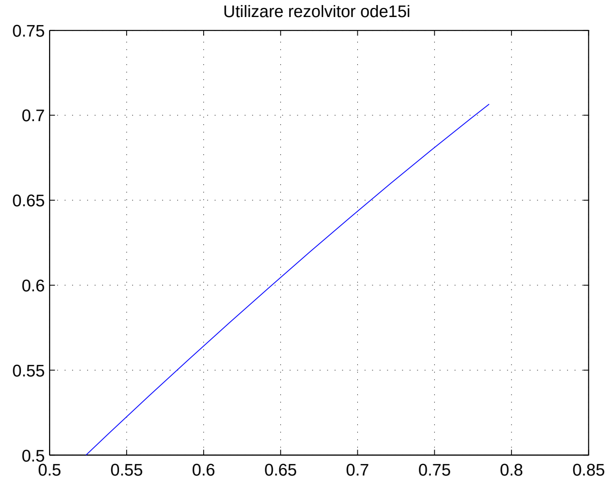
Cod Matlab
Câmpul vectorial este definit de**:**

```
function z= implic(t,y,yp )
z=yp^2+y^2-1;
end
```

În spaţiul de lucru a Matlabului tastăm:

```
>>tspan=[pi/6,pi/4];
>>[t,y]=ode15i(@implic,tspan,1/2,sqrt(3)/2);
>>plot(t,y)
>>grid
>>title('Utilizare rezolvitor ode15i ')
```

Graficul rezultat este:



10.3. Sisteme dinamice

10.3.1. Prezentare generală

În cele ce urmează vom considera un sistem dinamic definit de:

$$\frac{du}{dt} = f(u), u(0) = U, \ U \in \Re^{p}, \qquad \text{(PN)}$$

$U(t) \in \Re^p$ este o apllicaţie cu valori vectoriale u: $\Re \rightarrow \Re^p$, iar $f \in C(\Re^p, \Re^n)$ este neliniară şi nu depinde explicit de timp şi este de fapt **câmpul vectorial** care defineşte problema.
În cele ce urmează nu soluţiile acestui sistem ne interesează, ci orbitele sau traiectoriile punctelor $(u_1(t),u_2(t), ...u_p(t))^T$.
Vom da în continuare o serie de definiţii utile pentru a înţelege acest tip de sisteme de ecuaţii.

Definiţie
Problema Cauchy (PN) se spune că defineşte un **sistem dinamic** pe o submulţime $E \subseteq \Re^p$, dacă pentru fiecare dată iniţială $U \in E$, există o soluţie unică a acestei problemei şi care este definită pentru $t \in [0,\infty)$ şi rămâne inclusă în E pentru aceste momente de timp.
Teoreme de existenţă şi unicitate se găsesc de exemplu în lucrarea [9 pp. 22:25].

Definiţie
Pentru sistemul dinamic generat de (PN) definim **semigrupul de evoluţie** ca fiind aplicaţia $S(t): E \rightarrow E$ astfel încât $u(t)=S(t)u(0)$ pentru toţi $t \geq 0$. Acest operator are proprietăţile :

a) $u(t+s)=S(s)u(t)=S(t)u(s)$, pentru $t,s \geq 0$,
b) $S(0)=I_p$, I_p fiind operatorul identic în $\Re^p$.

Definiţie
Pentru orice submulţime B Í $\Re^p$, definim **acţiunea** semigrupului de evoluţie asupra acesteia prin:

S(t)B:=ÈS(t)U, U∈B.

Definiţie
Pentru sistemul dinamic S(t)(.) corespunzător (PN) definim **orbita pozitivă** respectiv **orbita înainte** pentru data U ca fiind mulţimea :

$\Gamma^+=\{ S(t)(U): t \geq 0\}$, iar **orbita negativă** $\Gamma^-=\{ S(t)(U): t £ 0\}$.

Definiţie
Un punct $x \in \Re^n$ se numeşte **punct ω limită** corespunzător datei U pentru un sistem dinamic , dacă există un şir t_i, $i=1,2,...\infty$, astfel ca $S(t_i)U \rightarrow x$, pentru $i \rightarrow \infty$. Mulţimea acestor puncte pentru o dată iniţială U se numeşte **mulţimea ω limită alui U.**
Folosind orbitele negative, când acestea există se introduc în mod analog **punctele α limită şi mulţimi α limită.**

Pentru sisteme dinamice unidimensionale singurele mulţimi limită pentru soluţii mărginite, sunt puncte critice. În cazul celor bidimensionale, tot pentru soluţii mărginite, pe lângă punctele critice ca mulţimi limită pot apăre şi soluţii periodice (cicluri limită).
Teorema lui Poincare-Bendixon [33]
Considerăm că pentru sistemul dinamic (PN) cu p=2 şi presupunem că pentru o dată iniţială $U \in \Re^2$ există $K > 0$ astfel încât :

$$\|S(t)U\| \leq K, \text{ pentru toţi } t \geq 0$$

Dacă ω (U) nu conţine nici un punct critic, atunci această mulţime este o soluţie periodică numită **ciclu limită.**

Definiţie
O aplicaţie $p(t)\in C^1(\Re\times\Re)$ se numeşte o soluţie **periodică cu perioada T** a problemei (PN), dacă este o soluţie a acesteia şi $p(t)=p(t+T)$, $\forall\ t\in\Re$, iar $p(t)\neq p(t+s)$, $s\in(0,T)$.

Corolar
Sistemele dinamice liniare nu pot avea soluţii periodice, deci nu există cicluri limită.

În studiul unui sistem dinamic, un pas esenţial este determinarea punctelor critice, adică a zerourilor sistemului algebric $f(x,y)=0$. Dacă dorim să explorăm dinamica unui astfel de sistem care depinde de un parametru folosim observaţia că punctele critice al acestui sistem sunt mulţimile α şi ω corespunzătoare acestora. Astfel putem obţine valori aproximative ale punctelor critice trasând câteva orbite selectate folosind paşi de integrare în timp pozitivi sau negativi.
În anumite situaţii aplicarea repetată poate fi destul de ineficientă.

10.3.2. Aplicaţii ale sistemelor dinamice în rezolvarea unui sistem neliniar

C.I Gheorghiu în lucrarea [9, pp:129-130] sugerează un procedeu practic şi sistematic de determinare a zerourilor lui $f(x,\mu)=0$ folosind sisteme dinamice.

Algoritm

- **Pasul 1**

Fixăm un $\mu=\mu_0$ şi folosim ecuaţia scalară $x'=f(x,\mu_0)$ cu diferite date iniţiale pentru a găsii mulţimea zerourilor lui
$f(x,\mu_0)=0$ pe care o notăm cu $E\mu_0$.

- **Pasul 2**

Pentru fiecare $x_0\in E\mu_0$ calculăm numeric soluţia problemei:

$$\mu'=-\frac{\partial f}{\partial x}(x,\mu),$$
$$x'=\frac{\partial f}{\partial \mu}(x,\mu)$$
$$\mu(0)=\mu_0$$
$$x(0)=x_0$$

integrând înainte şi înapoi folosind paşi de timp pozitivi respectiv negativi.

- **Pasul 3**

Se repetă, dacă este necesar Pasul 1pentru alte valori ale parametrului μ deoarece pot exista componente ale curbelor zerourilor lui $f(x,\mu)$ care nu intersectează dreapta $\mu=\mu_0$.

10.3.3. Bifurcaţii Hopf pentru prototipul unei ecuaţii diferenţiale cu argument întârziat

Argumentul întârziat apare în sistemele cu feedback din ştiinţe medicale şi inginerie. Feedbackul cu argument întârziat poate avea o influenţă considerabilă asupra comportării calitative a acestor sisteme [11].
Prototipul unei ecuaţii diferenţiale cu argument întârziat este descris astfel:

$$y'(t)=f(t,y(t),y(t-\tau_1),y(t-\tau_2),\ldots,y(t-\tau_k)) \quad \text{(PEDR)}$$

unde $0<\tau_1<\tau_2<\ldots<\tau_k$ sunt constante positive numite întârzieri.

Prezenţa unui parametru implică în multe cazuri comportări diferite ale traiectoriilor unui astfel de sistem, ceea ce în literatura de specialitate sunt cunoscute ca **bifurcaţii Hopf.**

Bifurcaţii dintr-o valoare proprie simplă

Forma normală pentru bifurcaţia de tip flip

Vom trata următorul sistemul dinamic unidimensional depinzând de un parametru:

$$x \to (1+\alpha)x + x^3 = f(x,\alpha)$$

Pentru orice α sistemul are un punct fix $x_0 = 0$, cu multiplu $\mu = -(1+\alpha)$. Punctul fix este stabil, dacă $\alpha < 0$ şi instabil dacă $\alpha > 0$. Dacă $\alpha = 0$ punctul fix $x_0 = 0$ nu este hiperbolic deoarece $\mu = f_x(0,0) = -1$, deci în acest caz apare o bifurcaţie flip.

Forma normală pentru bifurcaţia de tip fold.

Fie sistemul dinamic unidimensional depinzând de un parametru:

$$x \to \alpha + x + x^2 = f(x,\alpha)$$

Pentru $\alpha=0$ sistemul dinamic are un punct fix $x_0 = 0$, cu multiplu $\mu = f_x(0,0) = 1$. Pentru $\alpha < 0$ sistemul are două puncte fixe : $x_1 = -\sqrt{-\alpha}$ este stabil, iar $x_2 = \sqrt{-\alpha}$ este instabil.

În cazul unui sistem dinamic discret neliniar (PEDR) care depinde de un parametru real μ şi dorim să vedem cum este influenţată dinamica lui de variaţiile parametrului μ procedăm astfel:

- Considerăm Jacobianul aplicaţiei depinzând de de un parametru în vecinătatea punctelor fixe.
- Dacă o valoare proprie a acestuia trece prin 1, când parametrul variază apar unul sau mai multe puncte fixe,
- Dacă în schimb o valoare proprie trece prin -1 apare o soluţie periodică de perioadă 2.

Există două metode practice care pot duce la înţelegerea mai profundă a comportării unor astfel de sisteme dinamice :

- Diagrame pânză de paianjen
- Metoda seriilor temporale , această metodă înseamnă vizualizarea lor în sistemul de coordonate nOx_n.

Exemplu 1 [9,p:130]

Să se studieze diagrama de bifurcaţie de tip ***'flip'*** pentru ecuaţia:

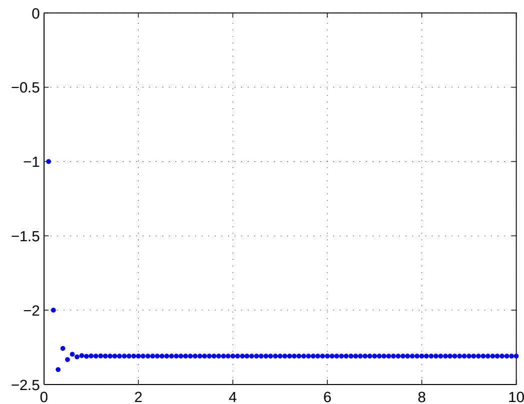
$$x' = \mu + x - x^3, \mu \in \Re.$$

Codul Matlab

```
%câmpul vectorial introdus
function yp = bifur(t,y )
%Parametrii
lambda=-10;
%Ecuatia
yp=lambda+y(1)-y(1)^3;
return
% functia
function [ t,y ] = ecdifn( odefun,tspan,y,Nh,varargin )
%Metoda lui Euler simplectică
h=(tspan(2)-tspan(1))/Nh;
tt=linspace(tspan(1),tspan(2),Nh+1);
for t=tt(1:end-1)
        y=[y,y(:,end)+h*feval(odefun,t,y(:,end),varargin{:})];
end
t=tt;
hold off
```

```
v=[-10 10 -10 10];
axis(v);
axis square;
drawnow;
u=y(1,:);% reprezentăm grafic
plot(t,u,'.', 'markersize',10)
grid on
return
%Apelul din linia de comanda
[t,y]=ecdifn('bifur',[0,10],0.,100);
```

În urma executării acestui program pentru $\lambda = -10$ va rezulta următorul grafic:

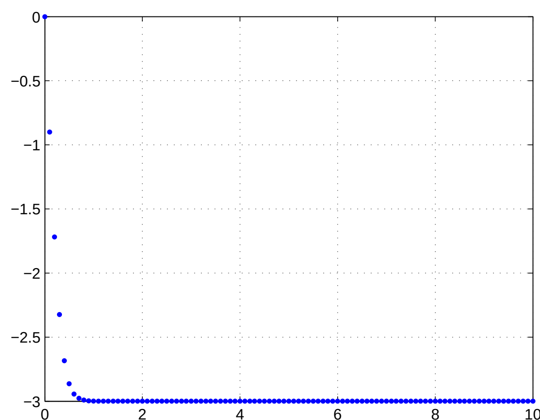


Este interesant de observat **instabilitatea acestei metode,** soluţia numerică prezintă acel ***zig-zag*** înainte de a converge la -2.3089.

Exemplul 2 [9, pag. 130]
Să se studieze diagrama de bifurcaţie de tip **fold** pentru ecuaţia:

$$x' = \mu + x^2 - 1;\ \mu = -8.$$

Folosind codul Matlab de la **exemplul 1** se obţine graficul:

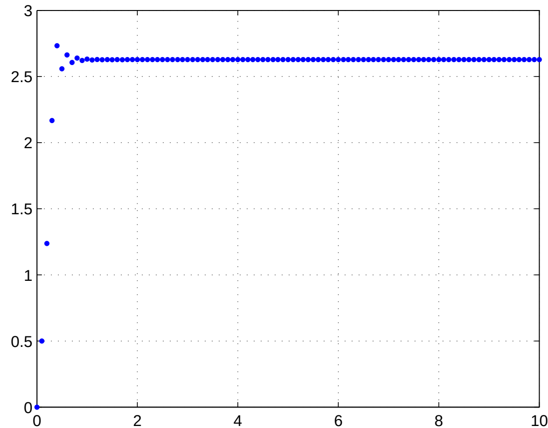


Aceasta este o bifurcaţie de **tip şa.**

Exemplul 3 [9, pag.135]
Să se studieze diagrama de bifurcaţie de tip **flip** pentru următoarea ecuaţie:

$$x' = \mu + \mu x - x^3; \ \mu = 5;$$

Folosind codul Matlab de la exemplul 1 se obţine graficul:



10.3.4. Aplicaţii ale sistemelor dinamice

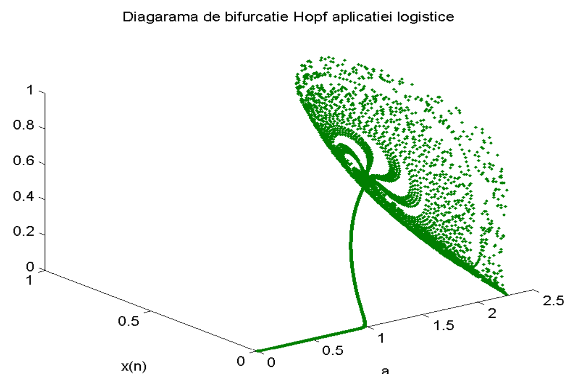
A) Metoda diagramei pânză de paianjen [11]

Exemplu 4
Folosind ***metoda diagrame pânză de paianjen***, să se studieze gradul de împrăştiere a unui proiectil dat de ecuaţia diferenţială:

$$x' = ax(1-x), \ x(0.1)=0; \ \text{pentru } a \in [0,2.5].$$

Cod Matalb
```
clear all
close all
itermax=100;
finalits=30; finits=itermax-(finalits-1);
for a=0:0.0025:2.5
    x=0.1;y=0;
    x0=x;y0=0;
    for n=2:itermax
        xn=a*x0*(1-y0);
        yn=x0;
        x=[x,xn];y=[y,yn];
        x0=xn;y0=yn;
    end
plot3(a*ones(finalits),x(finits:itermax),y(finits:itermax),'.',...
        'Markersize',5)
    hold on
end
fsize=10;
axis on
```

```
xlabel('a','Fontsize',fsize)
ylabel('x(n)','Fontsize',fsize)
title('Diagrama de bifurcatie Hopf a aplicatiei logistice')
```
În urma execuţiei acestui cod Matlab se obţine graficul:



B) Cicluri limită

Exemplul 6
Să se scrie un program Matlab pentru trasarea unor traiectorii din planul fazelor pentru modelul pradă-prădător cunoscut în literatura de specialitate ca **modelul Lotka-Voltera** şi dat de sistemul :

$$y_1' = y_1 - a*y_1*y_2$$
$$y_2' = -y_2 + b*y_1*y_2$$

Un studiu analitic şi numeric detaliat precum şi aplicaţii în modelarea matematică în medicină (interacţiunea parzit- gazdă) se poate găsii în lucrarea [23, pag:229-236].

Codul Matlab
% program care implemeteaza metoda Euler-progresiva
```
function [ t,y ] = ecdifn( odefun,tspan,y,Nh,varargin )
%Metoda Euler progresiva
h=(tspan(2)-tspan(1))/Nh;
tt=linspace(tspan(1),tspan(2),Nh+1);
for t=tt(1:end-1)
    y=[y,y(:,end)+h*feval(odefun,t,y(:,end),varargin{:})];
end
t=tt;
hold off
v=[-10 10 -10 10];
axis(v);
axis square;
drawnow;
u=y(1,:);
v=y(2,:);
plot(u,v,'.', 'markersize',15)
title('O traiectorie a modelului L-V a=2,b=3')
grid on
return
```

Câmpul vectorial corespunzător

```
function yp = bifur(t,y )
%Parametrii
a=2.;
b=3.;
%Ecuatiile
yp=[y(1)-a*y(1)*y(2); -y(2)+b*y(1)*y(2)];
return
```

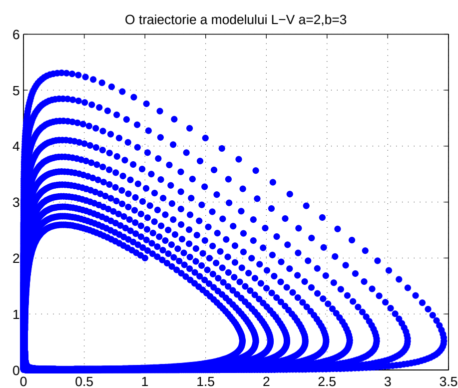
În urma apelului din linia de comanda:

`>>[t,y]=ecdifn('bifur',[t0,tf],[y1;y2],Ni);`

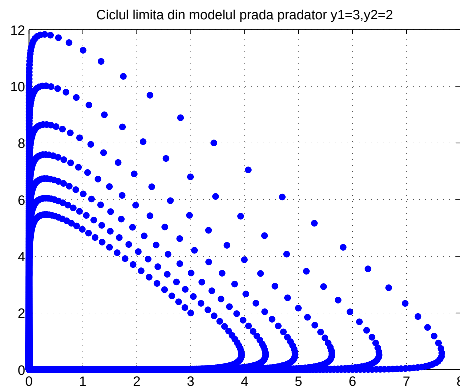
t_0, și t_f sunt momentele în care se integrează, iar y_1 și y_2 sunt datele inițiale, N_h sunt numărul pașilor de integrare

`>>[t,y]=ecdifn('bifur',[0,120],[1;2],10000);`

Pentru $y_1=1, y_2=2$ se obține graficul:



Pentru $y_1=3, y_2=2$ se obține graficul:



Observație

Metoda ***Euler*** explicită ***NU este capabilă*** să realizeze figura de mai sus datorită instabilității sale.

Exemplul 7

Să se scrie un program Matlab pentru trasarea unor traiectorii din planul fazelor pentru modelul ***Brusselator*** [26].

$$y_1' = a-(b+1)y_1-y_1\hat{}2*y_2$$
$$y_2' = by_1- y_1\hat{}2*y_2$$

Să se arate că acesta prezintă o **bifurcaţie Hopf.**

Soluţie

Unicul punct critic al acestui sistem este $x=a, y=b/a$. Liniarizând în jurul acestuia, ecuaţia pentru valorile proprii este:

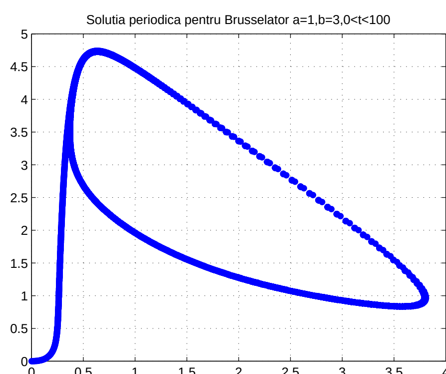
$$\lambda^2+\lambda(a^2-b+1)+a=0$$

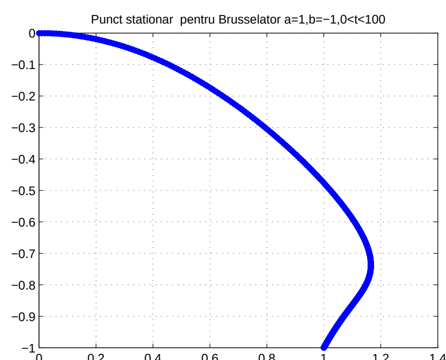
Bifurcaţia Hoph este posibilă numai dacă în planul parametrilor a,b un punct curent traversează parabola $b=a^2+1$.

Atunci valorile proprii devin $\pm$ ai.

Dacă de exemplu a=1, b=3 avem un ciclu limită, iar pentru a=1, b=1 punctul critic este staţionar şi nu mai există soluţii periodice.

Un studiu matematic detaliat al ciclurilor limită se pot găsii de exemplu în monografia [27] .





Exemplul 8

În lucrarea : ***Nonlinear models of reaction-diffusion in rivers vol IV , pp. 217-219 New Brunswick, NJ: IMACS, 1981, W.Ames şi E.Lohner*** studiază următoarea problemă care apare în studiul poluării apelor râurilor curgătoare:

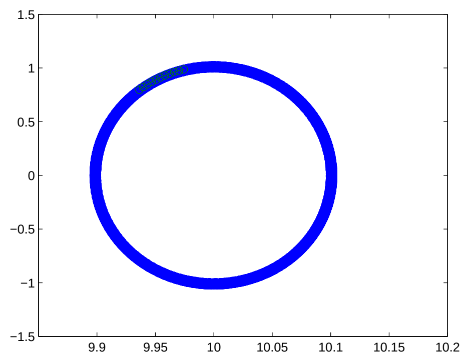
$$y_1'(x) = y_2(x)$$
$$y_2'(x) = 100*y_1(x)-10*y_1^2$$
$$y_1(0) = 1,\ y_1'(0) = \infty$$

Acesta este un sistem dinamic neliniar cu punctele fixe (0,0) şi (10,0) . Un studiu analitic simplu demonstrează că punctul fix (10,0) este un nod stabil, iar traiectoriile generează un ***ciclu limită.***

COD MATLAB

```
clear all;
close all;
tspan=[0,100];
y0=[9.9,0.001];
opts = odeset('OutputFcn',@odephas2,'Refine',10);
a=10;b=10;
tic;
[t,y]=ode113(@Amesloh,tspan,y0,opts,a,b);
toc;
plot(y(:,1),y(:,2),'g*',[9.9,0.001],'k*');
axis([0.0001 20 0.0001 5]);
grid on
xlabel('x(t)'), ylabel('h(t)')
```

Se poate observa în graficul următor prezenţa unui astfel de ciclu limită în jurul punctului critic (10,0).



10.4. Aplicaţii ale sistemelor de ecuaţii diferenţiale cu argument întârziat în medicină

10.4.1. Modelul Corvin, Sarafyan şi Thomson

Vom considera modelul **Corvin, Sarafyan, & Thomson 1997,** care descrie cum se transmite virusul HIV la o populaţie omogenă. În acest model **x** reprezintă numărul indivizilor succeptibili de infectare, **y** numărul efectiv de indivizi efectiv infestaţi , **λ** este rata de imbolnăvire pe o unitate de timp , **D** durata relaţiei , **c** este numărul contactelor sexuale de lungă durată între parteneri, **G** este rata de însănătoşire a celor infectaţi, **n** este numărul indivizilor populaţiei omogene.

În scrierea acestor coduri Matlab am folosit ideile prezentate în lucrarea [25, pag 213-230]:
Pentru $t \le D$ ecuațiile sunt:

$$x'(t)=-x(t)\lambda(t)$$

$$n=x(t)+y(t)$$

$$\lambda(t)=\frac{c}{n}\left\{\int_0^1\int_x^1 \exp(-G(t-w)x(w)\lambda(w)dwds+\int_0^t \exp\big(-G(t-s)\big)\,y(s)ds\right\}$$

Iar pentru $D \le t$:

$$x'(t)=-x(t)\lambda(t)$$

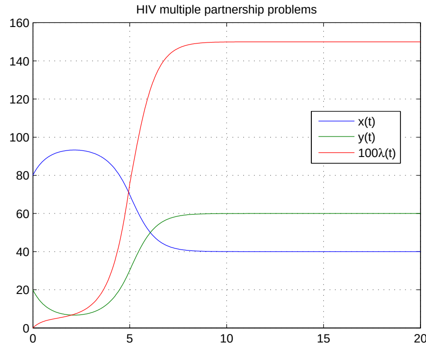
$$n=x(t)+y(t)$$

$$\lambda(t)=\frac{c}{n}\left\{\int_{t-D}^t\int_s^t \exp(-G(t-w)x(w)\lambda(w)dwds+\int_{t-D}^t \exp\big(-G(t-s)\big)\,y(s)ds\right\}$$

Codul Matlab este:

```
function sol = HIV
global c D G n
c=0.5; D=5; G=1; n=100;
y0=[0.8*n; 0.2*n; 0; 0; 0; 0];
options = ddeset('RelTol',1e-5);
sol=dde23(@odes,[],y0,[0, D],options);
sol=dde23(@ddes,D,sol,[D,4*D],options);
plot(sol.x,[sol.y(1:2,:); 100*sol.y(3,:)]);
grid
title('HIV multiple partnership problems');
legend ('x(t)','y(t)','100\lambda(t)',0);
function dydt=odes(t,y,Z)
global c D G n
dydt=zeros(6,1);
dydt(1)=-y(1)*y(3)+G*y(2);
dydt(2)=-dydt(1);
dydt(4)=exp(G*t)*y(2);
dydt(5)=t*exp(G*t)*y(1)*y(3);
dydt(6)=exp(G*t)*y(1)*y(3);
dydt(3)=(c/n)*exp(-G*t)*((dydt(4)+dydt(5))-G*(y(4)+y(5)));
function dydt=ddes(t,y,Z)
%folosim rezolvitorul ddes deoarece există ami multi parametrii
global c D G n
dydt=zeros(6,1);
dydt(1)=-y(1)*y(3)+G*y(2);
dydt(2)=-dydt(1);
dydt(4)=exp(G*t)*y(2)-exp(G*(t-D))*Z(2);
dydt(5)=D*exp(G*t)*y(1)*y(3)-y(6);
dydt(6)=exp(G*t)*y(1)*y(3)-exp(G*(t-D))*Z(1)*Z(3);
dydt(3)=(c/n)*exp(-G*t)*((dydt(4)+dydt(5))-G*(y(4)+y(5)));
```

În urma execuției acestui program obținem rezultatul:



10.4.2 Modelul lui Genik şi van den Driessche

În lucrarea " ***An Epidemic Model with Recruiment-Death Demographics and Discrete Delays" Differential Equations with Application to Biology pp237-249, Providence, R.I:American Mathematical Society 1999, Genik & van den Driessche*** consideră că numărul total al unei populaţiei N(t) poate fi împărţit în patru submulţimi:

- S(t) suspecţi de îmbolnăvire,
- E(t) expuşi la îmbolnăvire,
- I(t) infectaţi,
- R(t) recuperaţi în urma unui tratament adecvat.

Deci :

`N(t)= S(t)+E(t)+I(t)+ R(t)`

Cei cinci parametrii ai acestui model pentru virusul *Pasteurella muris* sunt: A=0.330 rata naşterii, d=0.006 rata de mortalitate, rata de îmbolnăvire în urma unui contact individual λ = 0.308, rata de însănătoşire γ = 0.040, iar = 0.060 reprezintă rata de maortalitate excesivă. Există două valori care pot întârzia aceste rate : imunitatea τ= 42.0 şi timpul de expunere , ω = 0.15.

Deci acest model se poate descrie sub forma unui sistem de ecuaţii diferenţiale cu argument întârziat astfel:

$$\frac{dS(t)}{dt} = A - dS(t) - \lambda\frac{S(t)I(t)}{N(t)} + \gamma I(t-\tau)\exp(-d\tau)$$

$$\frac{dE(t)}{dt} = \lambda\frac{S(t)I(t)}{N(t)} - \lambda\frac{S(t-\omega)I(t-\omega)}{N(t-\omega)}\exp(-d\tau) - dE(t)$$

$$\frac{dI(t)}{dt} = \lambda\frac{S(t-\omega)I(t-\omega)}{N(t-\omega)}\exp(-d\tau) - (\gamma+\varepsilon+d)I(t)$$

$$\frac{dR(t)}{dt} = \gamma\, I(t) - \gamma\, I(t-\tau)\exp(-d\tau) - dR(t)$$

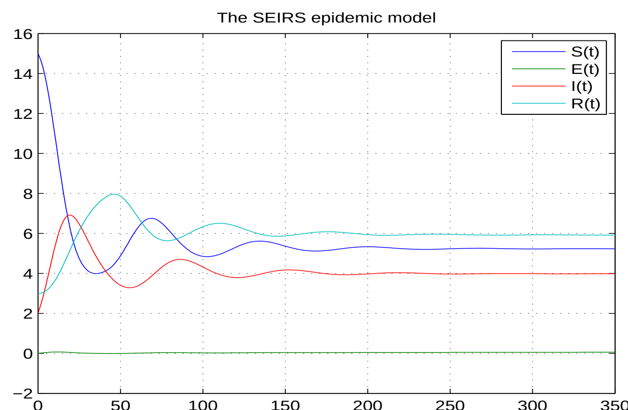
Vom rezolva acest sistem pe [0, 350], iar ca valori iniţiale vom lua `S(t)=15, E(t)= 0, I(t)= 2, R(t)=3 pentru t ≤ 0.`

Codul Matlab este:

```
function sol = genik
global tau omega
```

```
tau=42.0; omega=0.15;
sol=dde23(@ddes,[tau,omega], [15; 0; 2; 3],[0,350]);
plot(sol.x, sol.y)
grid
title('The SEIRS epidemic model');
legend ('S(t)', 'E(t)','I(t)','R(t)');
%=============
    function dydt=ddes(t,y,Z)
        global tau omega
        %parametrii
        A=0.330; d=0.006; lambda=0.308;
        gamma=0.040; epsilon= 0.060;
        S=y(1); E=y(2); I=y(3); R=y(4);
        Itau=Z(3,1);
        Somega=Z(1,2);Eomega=Z(2,2);
        Iomega=Z(3,2); Romega=Z(4,2);
        Noft= S+E+I+R;
        Nomega=Somega+Eomega+Iomega+Romega;
        dSdt=A-d*S-lambda*((S*I)/Noft)+gamma*Itau*exp(-d*tau);
        dEdt=lambda*((S*I)/Noft)-...
            lambda*((Somega*Iomega)/Nomega)*exp(-d*omega)-d*E;
        dIdt=lambda*((Somega*Iomega)/Nomega)*exp(-d*omega)...
            - (gamma+epsilon+d)*I;
        dRdt=gamma*I-gamma*Itau*exp(-d*tau)-d*R;
        dydt=[dSdt; dEdt; dIdt; dRdt];
```

În urma execuţiei acestui script obţinem rezultatul:



10.4.3. Modelul Cardiovascular

Următorul **model cardiovascular** datorat lui ***Ottensen J.T.***, şi prezentat în lucrarea**: Modelling the dynamical baroreflex-feedback control,** ***Mathematical and computer modelling, 31: 167-173, 2000***, implică următoarele funcţii $P_a(t)=y_1(t)$, presiunea arterială, $P_v(t)=y_2(t)$, presiunea venoasă, pulsul inimii $H(t)=y_3(t)$.
El sugerează rezolvarea acestui model folosind următorul sistem de ecuaţii diferenţiale cu argument întârziat :

$$y_1'(t) = -\frac{1}{R*c_a} y_1(t) + \frac{1}{R*c_a} y_2(t) + \frac{1}{c_a} \text{Vstr}\, y_3(t)$$

$$y_2'(t) = -\frac{1}{R*c_a} y_1(t) - \left(\frac{1}{R*c_v} + \frac{1}{r*c_v}\right) y_2(t)$$

$$y_2'(t) = f(T_s, T_p)$$

unde :

$$f(T_s, T_p) = \frac{\alpha_H}{1+\gamma_H T_p} T_s - \beta_H T_p$$

Ottesen studiază condiţiile în care întârzierea cauzează diferenţe calitative în soluție şi în special oscilaţii în $P_a(t)$.

Pentru $t \leq 0$ soluţia este constantă.

$$y_1(t) = P_0$$

$$Y_2(t) = (r/r+R) P_0$$

$$Y_3(t) = (1/r+R)(P_0/Vstr)$$

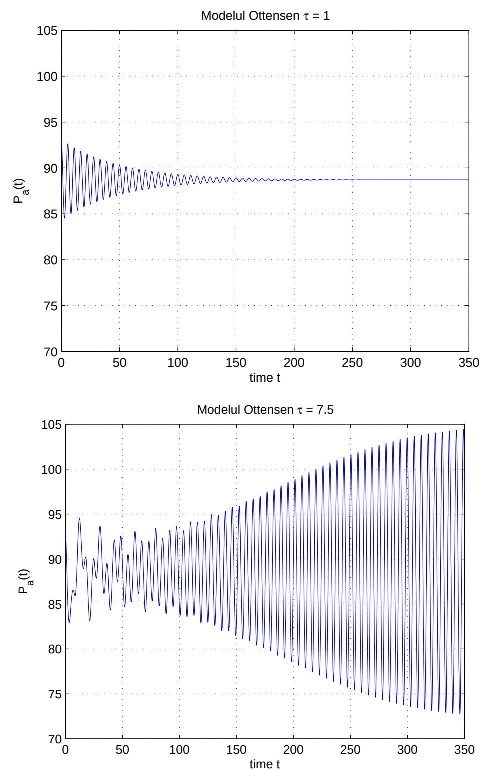
Dacă $c_a = 1.55$, $c_v = 519$, $R = 1.05$, $r = 0.068$, $\alpha_0 = \alpha_s = \alpha_p = 93$, $\alpha_H = 0.84$, $\beta_0 = \beta_s = \beta_p = 7$, $\beta_H = 1.17$, $\gamma_H = 0$ $V_{str} = 67.9$, $P_0 = 93$
$\tau \in \{1\ 7.5\}$

Folosind următorul **cod Matlab** se obţin rezultatele:

```
function sol = ottensen
global R r Vstr
R = 1.05;
r = 0.068;
Vstr = 67.9;
P0     = 93;
Paval = P0;
Pvval = (1 / (1 + R/r)) * P0;
Hval  = (1 / (R * Vstr)) * (1 / (1 + r/R)) * P0;
history = [Paval; Pvval; Hval];
opts = ddeset('RelTol',1e-5,'AbsTol',1e-8);
for tau = [1 7.5]
   sol = dde23(@ddes,tau,history,[0, 350],opts);
   figure
   plot(sol.x,sol.y(1,:))
   title(['Modelul Ottensen' ...
          ' \tau = ',num2str(tau)])
   xlabel('time t')
   ylabel('P_a(t)')
   axis([0 350 70 105])
end

%========================================================
function v = ddes(t,y,Z)
global R r Vstr
ca = 1.55;
cv = 519;
gammaH = 0;
alpha0 = 93;
alphas = 93;
alphap = 93;
alphaH = 0.84;
beta0  = 7;
betas  = 7;
```

```
betap  = 7;
betaH  = 1.17;
ylag = Z(:,1);
Patau = ylag(1);
Paoft = y(1);
Pvoft = y(2);
Hoft  = y(3);
dPadt = - (1 / (ca * R)) * Paoft + (1/(ca * R)) * Pvoft ...
        + (1/ca) * Vstr * Hoft;
dPvdt = (1 / (cv * R)) * Paoft                           ...
        - ( 1 / (cv * R) + 1 / (cv * r) ) * Pvoft;
Ts = 1 / ( 1 + (Patau / alphas)^betas );
Tp = 1 / ( 1 + (alphap / Paoft)^betap );
dHdt = (alphaH * Ts) / (1 + gammaH * Tp) - betaH * Tp;
v = [ dPadt; dPvdt; dHdt ];
```

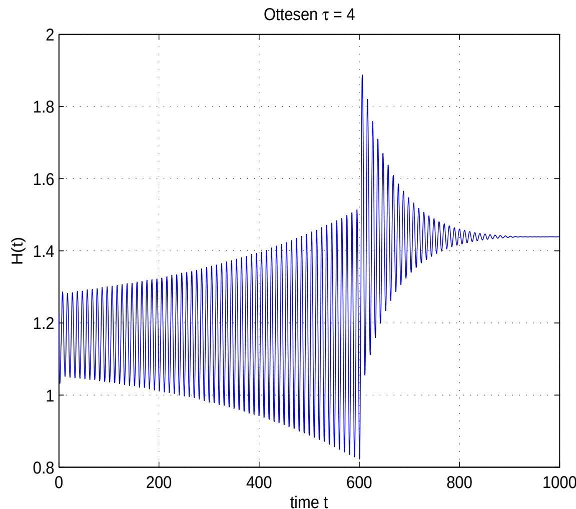


Dacă `t=600`, presiunea periferică `R` se reduce exponenţial de la valoarea constantă `R=1.05` la valoarea constantă `R=0.84` şi `τ=4` în intervalul de întârziere `[0,1000]`.

Folosind următorul program Matlab:

```
function sol = ottesen1
global r Vstr
R = 1.05; r = 0.068; Vstr = 67.9;
P0      = 93;
Paval   = P0;
Pvval   = (1 / (1 + R/r)) * P0;
Hval    = (1 / (R * Vstr)) * (1 / (1 + r/R)) * P0;
history = [Paval; Pvval; Hval];
tau = 4;
opts = ddeset('Jumps',600,'RelTol',1e-4,'AbsTol',1e-7);
sol = dde23(@ddes,tau,history,[0, 1000],opts);
plot(sol.x,sol.y(3,:))
grid
title(['Ottesen' ...
       ' \tau = ',num2str(tau)])
xlabel('time t')
ylabel('H(t)')
%===========================================================
function v = ddes(t,y,Z)
global r Vstr
ca = 1.55; cv = 519; gammaH = 0;
alpha0 = 93; alphas = 93; alphap = 93; alphaH = 0.84;
beta0  = 7; betas  = 7; betap  = 7; betaH  = 1.17;
if t <= 600
   R = 1.05;
else
   R = 0.21 * exp(600-t) + 0.84;
end
ylag = Z(:,1);
Patau = ylag(1);
Paoft = y(1);
Pvoft = y(2);
Hoft  = y(3);
dPadt = - (1 / (ca * R)) * Paoft + (1/(ca * R)) * Pvoft ...
        + (1/ca) * Vstr * Hoft;
dPvdt = (1 / (cv * R)) * Paoft                          ...
        - ( 1 / (cv * R) + 1 / (cv * r) ) * Pvoft;
Ts = 1 / ( 1 + (Patau / alphas)^betas );
Tp = 1 / ( 1 + (alphap / Paoft)^betap );
dHdt = (alphaH * Ts) / (1 + gammaH * Tp) - betaH * Tp;
v = [ dPadt; dPvdt; dHdt ];
```

pentru $\tau = 4$ se obține rezultatul:



10.4.4. Modelul lui Plant

În lucrarea ***Biological Delay Systems: Linear Stability Theory,*** *Cambridge University Press 1989,* **N.MacDonald** propune următorul sistem de ecuaţii diferenţiale cu argument întârziat care modelează matematic interacţiunea dintre doi neuroni din sinapse cunoscut ca: ***Plant's neuron interaction model*** model folosit în studiul reţelelor Neuronale.

$$y_1'(t) = -y_1(t) - y_2(t) - \frac{y_{1(t)}^3}{3} + m(y_1(t-\tau) - y_{1,0})$$
$$y_2'(t) = r(y_1(t) + a - by_2(t))$$

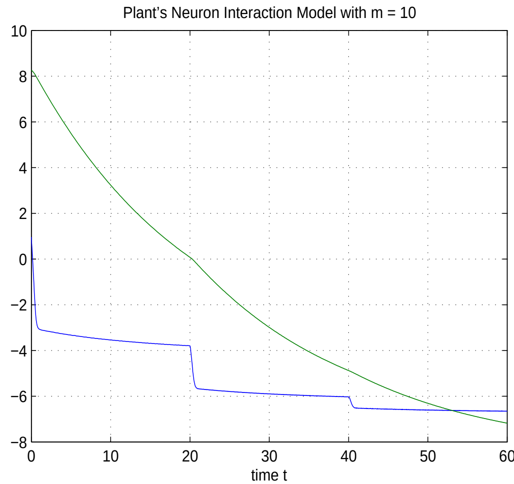
unde $y_{1,0}$ este prima componentă a soluţiei de echilibru ($y_{1,0}$, $y_{2,0}$) .

Dacă parametrii: $a = 0.8$, $b = 0.7$, $r = 0.08$, $\tau = 20$, $m = 10$, şi rezolvăm acest sistem de ecuaţii diferenţiale cu argument întârziat în intervalul: [0, 60] obţinem rezultatele:

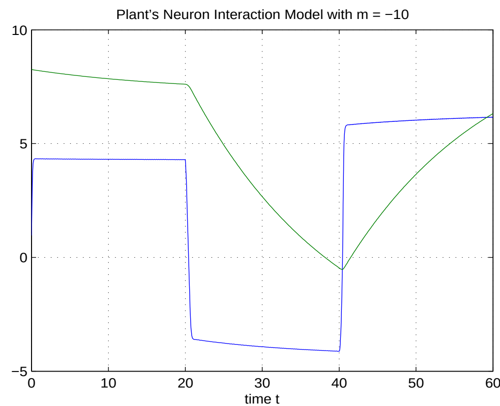
$$y1(t) = 0,4 \quad y_{1,0}$$

$$y_2(t) = 1,8 \quad y_{2,0}$$

Alegem $y_{1,0}$ = -1.22764016121492, şi vom rezolva cu ajutorul Matlab-ului această problemă pentru $\tau = 20$, separat pentru m = 10 şi m = -10.
Pentru m = 10 obţinem graficul:



Iar pentru m = -10 obţinem graficul:



10.4.5. Modelul matematic al răspunsului imunitar

Sistemul imunitar este format dintr-o mulţime de celule şi molecule care conţine cel puţin 10^6 clone ale celulelor ce acţionează între ele. Sistemul imunitar depinde de sistemul nervos şi de sistemul endocrin . Există în permanenţă o competiţie între sistemul imunitar şi agenţii infecţioşi [34].

Modelul răspunsului imunitar asupra unui agent infecţios descrie rezultatul competiţiei dintre cele două sisteme, considerând la un moment $t \in \Re$ o caracteristică a unui agent infecţios ce cuantifică răspunsul sistemului imunitar (microorganisme, macromolecule, celule de tumori imunogene) ca fiind $x(t)$ şi concentraţia agenţilor imunitari cei mai importanţi (anticorpi, celule NK,T-citotoxice) ca fiind $y(t$. Efectul răspunsului imunitar la un

moment dat este dat de $-kx(t)y(t)$, $k>0$, rata medie de producere a celulelor infecţioase este de $rx(t)$, $r>0$.
Variaţia caracteristicii agentului infecţios este dată de:

$$x'(t)=rx(t)-kx(t)y(t) \qquad (1)$$

Interacţiunea dintre variaţia concentraţiei agenţilor imunitari (durata medie de viaţă a acestora) şi caracteristica agentului infecţios este descrisă cu ajutorul funcţiilor f,g: $\Re_+ \to \Re_+$ derivabile şi este dată de:

$$y'(t)=pf(ax(t)+(1-a)x(t-\tau))+sg(y(t))-y(t), \qquad (2)$$

unde $\tau > 0$, $p>0$, $s>0$, $a\in[0,1]$ şi $x(t-\tau)$ reprezintă caracteristica agentului infecţios din momentul $t-\tau$.
Considerarea funcţiilor f, g şi a argumentului întârziat τ corespunde din punct de vedere biologic răspunsului imunitar, unde :

$$f(x)=\frac{x^4}{1+x^4},\ g(y)=\frac{y^3}{1+y^3},$$
$$x(\theta)=\varphi(\theta),\ \theta\in[-\tau,0],\ y(0)=y_0.$$

Analiza modelului răspunsului imunitar cu argument întârziat, constă în stabilirea variaţiei în raport cu timpul a variabilelor de stare x(t), y(t), în vecinătatea unei stări normale a fenomenului prin existenţa unor valori constante a caracteristicilor agentului infecţios şi a concentraţiei agenţilor imunitari.
Deci din punct de vedere matematic trebuie determinat punctul de echilibru şi de efectuat un studiu al stabilităţii în jurul acestui punct. Următoarea teoremă ne ajută în acest sens.

Teoremă

Următoarele afirmaţii sânt echivalente:

- Dacă $0 < s < \frac{1}{y_0^2}+y_0$ şi $p > y_0-\frac{sy_0^3}{y_0^3+1}$ atunci valorile :

$x_0=\sqrt{\frac{y_0(1+y_0^3)-sy_0^3}{(p-y_0)(1+y_0^3)}}$, $y_0=\frac{r}{k}$ sunt coordonatele punctului de echilibru (punctul fix al sistemului (1)+(2)) (3).

- Dacă $\tau = 0$ şi $y_0 > \sqrt[3]{2}$ şi $s < \frac{1}{y_0^2}+y_0$ atunci punctul de echibru este asimptotic stabil (4).
- Dacă $\tau = 0$ şi $0 < y_0 < \sqrt[3]{2}$ şi $s < \frac{1}{3}\left(\frac{1}{y_0^2}+y_0\right)^3$, atunci punctul de echilibru (x_0, y_0)

este asimptotic stabil (5)

- Dacă au loc (3) sau (4), atunci pentru orice $\tau\in[0,\tau_c]$, punctul de echilibru (x_0, y_0) este asimptotic stabil unde

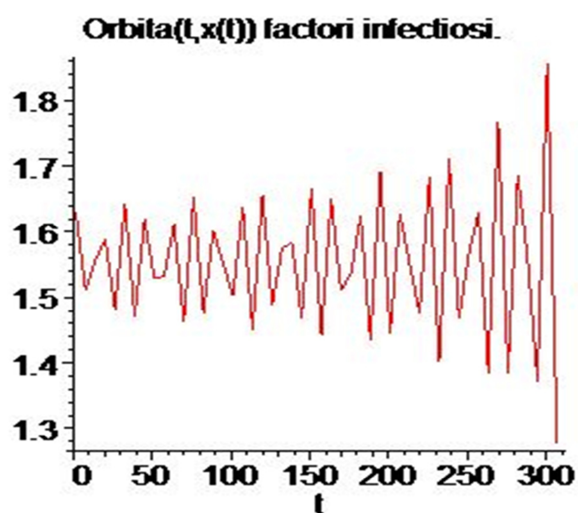
$$\tau_c=\frac{\left(1+y_0^3\right)^3-3sy_0^2}{(1-a)p\rho_1 kx_0\left(1+y_0^3\right)^2} \qquad (6)$$

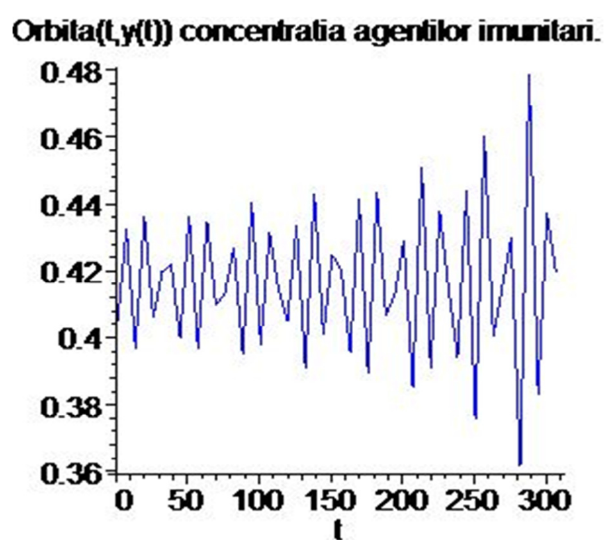
Pentru: $k=2.5$, $r=0.9$, $s=0.25$, $p=0.55$, $a=0.25$ suntem în cazul (6).

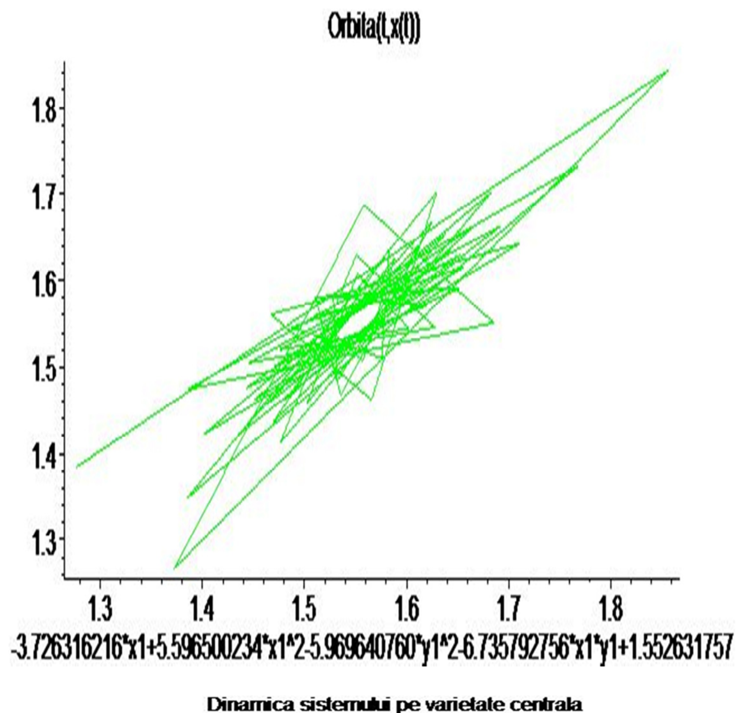
Scriind cod Matlab am obţinut punctul de echilibru $x_0= 1.147584637$, $y_0=0.36$ şi

$$\tau_c=1.144486805$$

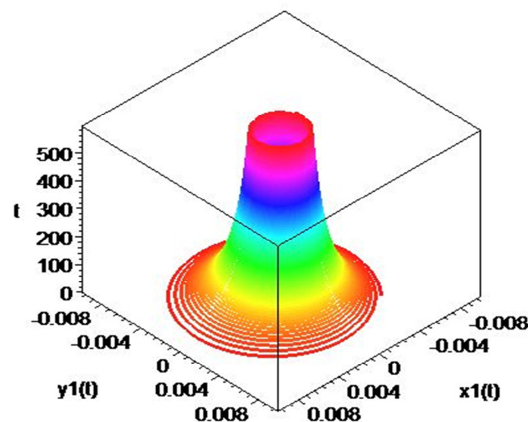
şi reprezentările grafice:


Orbita(t,x(t)) factori infectiosi.
1.8
1.7
1.6
1.5
1.4
1.3
0
50
100
150
200
250
300
t


Orbita(t,y(t)) concentratia agentilor imunitari.
0.48
0.46
0.44
0.42
0.4
0.38
0.36
0
50
100
150
200
250
300
t



Dinamica sistemului pe varietate centrala



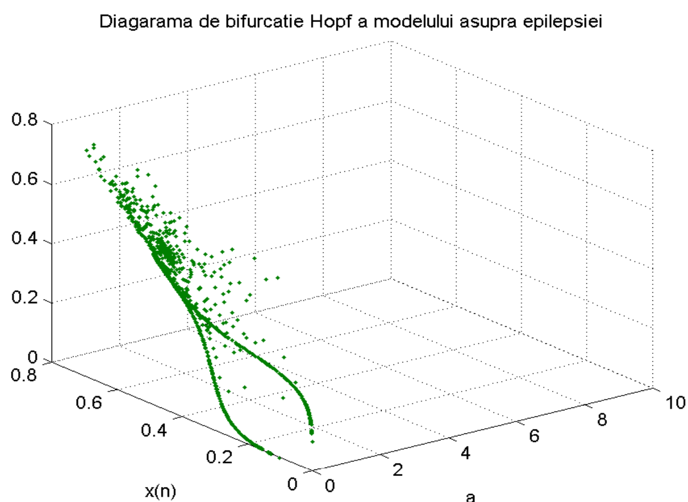
10.4.6. Modelul epilepsiei

Epilepsia (în greaca veche: ἐπίληψις, epílēpsis) este o boală a sistemului nervos central, caracterizată prin crize, accese convulsive intermitente, însoțite de pierderea cunoștinței, de halucinații și de alte tulburări psihice.
Cauza precisă a bolii nu este cunoscută, se presupun predispoziții ereditare, infecții cu inflamații acute sau intoxicații care pot să declanșeze accese epileptiforme. Accesele epileptice ar fi produse de o descărcare simultană a fluxului nervos la nivelul unor grupe de neuroni din creier.

Această diagnoză se poate pune cu ajutorul ajutorul **electroencefalogramei**. Medicația epilesiei este simptomatică, adică se caută combatrea simptomelor cu medicamente antispastice și relaxante. S-au făcut și intervenții chirurgicale pe creier, obținându-se rezultate îndoielnice.
Simptome epileptiforme asemănătoare s-au depistat și la animale.
Să se studieze gradul de împrăştiere al **modelului epilepsiei**, dat de ecuaţia:

```
x'= 4*a*x^3-6*a*x^2+(1+2*a)*x;  x(0.1)=0; pentru a∈[0,10].
```

Scriind cod Matalb obţine graficul:



10.4.7. Modelul matematic al testului IVGTT

Relaţia dinamică dintre glucoză şi insulină se stabileşte folosind testul de toleranţă al glucozei administrată intravenos cunoscut în literatura de specialitate ca testul IVGTT [12]. Trebuie menţionat ca această problemă este studiată la Universitatea Louisville – Kentucky U.S.A, iar eu în acesastă lucrare am preluat unele din aceste rezultate.
Variabile de stare ale modelului sunt: cantintatea de glucoză plasmă la momentul t, notată $G(t)$ [mg/t] şi concentraţia de insulină notată $I(t)$ [μUI/me] funcţii presupuse derivabile. Modelul matematic este dat de sistemul de ecuaţii diferenţiale :

$$G'(t)=-f(G(t))-g(G(t),I(t))+ b_7$$

$$I'(t)=-p(I(t))+q(G(t-\tau)), \qquad \text{(EDRI)}$$

Cu condiţia iniţială :

$$G(\theta)=\varphi(\theta),\ \theta\in[-\tau,0],\ I(0)=I_b + b_3 b_0,\ \tau > 0. \quad \text{(CI)}$$

$\varphi:[-\tau,0]\to\Re$ este o funcţie derivabilă cu $\varphi(0)= G_0+ b_0$, unde G_0, b_0, I_b, b_3, b_0 reprezintă parametrii
modelului şi se obţin prin analize de laborator. În continuare se consideră funcţiile:

$$f(x)=b_1 x,\ g(x,y)=\frac{b_4}{\propto x+1}xy,\ p(x)=b_2(x),q(x)=b_6 x, \quad \text{(FI)}$$

unde b_1, b_4, b_6 sunt numere positive ce se obţin din analize de laborator şi $\alpha \geq 0$ este un parametru .

Teoremă

Următoarele afirmaţii sunt echivalente:

- Sistemul de ecuaţii (EDRI) cu condiţia iniţială (CI) şi funcţiile (FI) are un unic punct de echibibru

$(I_0, G_0) \in \Re_+^2$ este dat de:

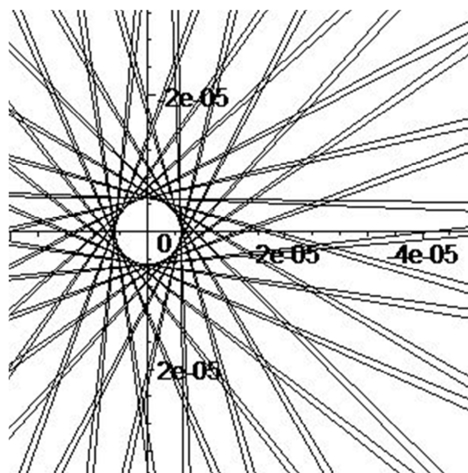
$$I_0 = \frac{b_6}{b_2} G_0,$$

$$G_0 = \frac{b_2(\alpha b_7 - b_1) + \sqrt{\Delta}}{2(b_1 b_2 \propto + b_4 b_7)}, \text{ unde } \Delta = b_2^2(b_1 - \alpha b_7)^2 + 4b_2b_7((b_1 b_2 \propto + b_4 b_7)$$

- Dacă $\tau = 0$ punctul de echilibru $(I_0, G_0) \in \Re_+^2$ este asymptotic stabil .
- Pentru orice $\tau \in [0, \tau_c]$ punctul de echibru $(I_0, G_0) \in \Re_+^2$ este asimptotic stabil dacă:

$$\tau_c = (a + b_2)/b_4 b_6$$

Utilizând un program Matlab pentru coeficienţii b_1=0.0002, b_0=209, b_2=0.042, b_3=1.64, b_7=0.68, α=0.001,
b_4=1.09E-0.4, b_5=23, b_6=0.033, se obţin următoarele rezultate:
Dacă în urma testului de glicemie se obţine G_0=133.87, în urma administrării cu insulină cu o medie zilnică (APIDRA) de I_0=104.69, se observă din graficul următor echilibrarea organismului.



Concluzie:

Din reprezentarea grafică de mai sus se observă prezenţa unui centru stabil (din punct de vedere al unui sistem dinamic), iar din punct de vedere al medicinei o echilibrare a raportului glicemie-insulină.

10.5. Sisteme de ecuaţii diferenţiale cu aplicaţii în dinamica fluidelor

În continuare vom da câteva exemple sugestive preluate din lucrarea de referinţă [3].

10.5.1. Căderea liberă a unui corp sferic

Să considerăm un corp sferic de masă *m* şi diametru *d* plasat la *t=0* în originea axei verticale descendente *Oz,* cu o viteză $v_0{}'$ în direcţia verticală care se deplasează sub acţiunea forţei gravitaţionale *mg* pe axa *Oz*.

La momentul *t* corpul se găseşte la distanţă *z(t)* de origine şi are viteza *v(t)* mărimi care verifică următorul sistem de ecuaţii diferenţiale:

$$\frac{dz}{dt} = v(t)$$

$$\frac{dv}{dt} = \frac{1}{A}\,[B - cv|v|c_d(v)]$$

Unde $A=1+\frac{\rho 2}{2}$, $B=(1-\rho 2)g$, $C=\frac{3\rho 2}{4d}$, iar $\rho 2 = \rho_f/\rho$ este raportul dintre densitatea fluidului şi cea a corpului,

c_d este coeficientul de rezistenţă la înaintare (*drag coefficient*) datorat vîscozităţii fluidului.

El depinde de numărul lui Reynolds R şi este dificil de evaluat analitic, determinându-se experimental sau utilizând codul Matlab următor :

Exemplu

Următoarele coduri Matlab sunt preluate din lucrarile [10], **[21].**

% functie care determina coeficientul Reynolds**:**

```
function cd = drag(Re)
if Re==0 cd=0;
elseif Re>=0 & Re<=1 cd=24/Re;
elseif Re>1 & Re<=400 cd=24/Re^0.646;
elseif Re>400 & Re<=3.e5 cd=0.5;
elseif Re>3.e5 & Re<=2.e6 cd=3.66e-4*Re^0.4275;
else cd=0.18;
end
```

Vom face următorul experiment cu o minge de ping-pong (cu densitatea egală cu a aerului) cu diametrul d=0.036m în apă , unde densitatea fluidului se poate lua $\rho_f = 1000\ kg/m^3$, $\nu = 1X10^{-6} m^2/s$, $\rho = 1.22 kg/m^3$, $g=9.81\ m/s^2$.

Vom scrie următoarele **programe Matlab**.

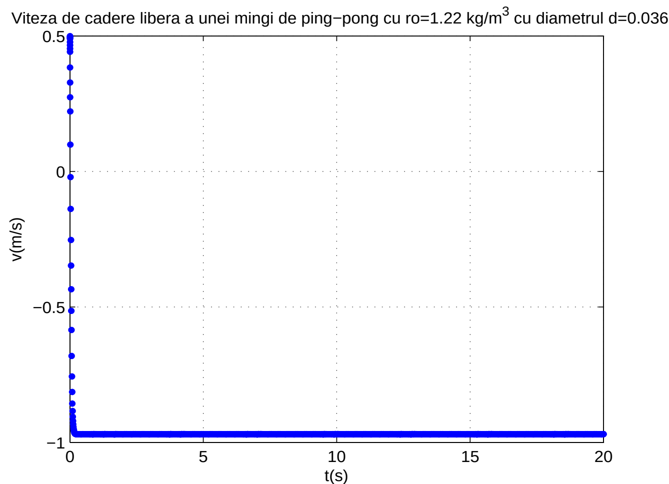
1.Câmpul vectorial

```
function yprim = corpsf(x,y)
global ro d rof nu;
yprim=zeros(2,1);
g=9.81;robar=rof/ro;
a=1+robar/2;b=(1-robar)*g;c=3*robar/4/d;
r=abs(y(2))*d/nu; cd=drag(r);
yprim(1)=y(2);
yprim(2)=(b-c*y(2)*abs(y(2))*cd)/a;
end
```

2 . Programul principal este:

```
function [t,x] = cadlib( ro,d,rof,nu,Tf,z0,v0 )
%cazul unei mingi de ping-pong
global ro d rof  nu;
ro=1.22;rof=1000;nu=0.000001;d=0.036;Tf=20;z0=0;
v0=0.5;%viteza de cadere libera
[t,x]=ode45('corpsf',[0,20],[z0,v0]);
plot(t,x(:,2),'.','MarkerSize',12);
grid on
xlabel('t(s)');ylabel('v(m/s)');
title('Viteza de cadere libera a unei mingi de ping-pong cu ro=1.22
kg/m^3 şi diametrul d=0.036')
end
```

În urma execuţiei acestui program rezultă graficul:



10.5.2 Mişcările verticale ale unei aripi de avion

Studiul vibraţiilor unei aripi de avion fixate elastic, în anumite condiţii de zbor, poate fi făcut tot prin simulare numerică, deoarece studiul analitic este dificil iar cel experimental costisitor.

Presupunem că greutatea aripii ***mg*** este suportată de un resort elastic de constantă *k*, echivalent cu fixarea elastică a aripii de avion. Dacă vântul lipseşte centrul de masă se află în starea de echilibru în *z=0* , iar arpa face un unghi α_0 cu orizontala. La deplasarea verticală a aripii, mişcarea este descrisă de cota *z(t)* şi viteza *v(t)* şi presupunem existenţa unui vânt orizontal de viteză *u.* Dacă luăm în considerare şi viteza *v,* unghiul de atac al aripii devine $\alpha=\alpha_0$-arctg(v/u), iar $\alpha\in[-18^o, 18^o]$. Dacă ţinem cont de forţa elastică generată de resort şi de cea ascensională generată de vânt, ecuaţiile mişcării devin:

$$\frac{dz}{dt} = v$$

$$m\frac{dv}{dt} = -kz + \frac{1}{2}\rho_f S c_1 u \sqrt[2]{u^2 + v^2}$$

unde ρ_f este densitatea aerului, S este proiecţia pe orizontală a ariei aripii , iar $c_l=2\pi\alpha$.

Aceste ecuaţii conţin mulţi parametrii, de aceea *T.Petrila şi D.Trif [21 pag360-362]* sugerează rezolvarea sistemului de mai sus în formă adimensională pentru a reduce numărul parametrilor. Ei sugerează folosirea unor mărimi de referinţă : timpul, lungimea şi viteza, alegând ca timp de referinţă perioada oscilaţiilor libere ale aripii $2\pi\sqrt{m/k}$, deformaţia aripii datorită greutăţii mg/k ca lungime de referinţă , iar raportul lor ca viteză de referinţă.
Definim mărimile adimensionale astfel:

$$T=t/2\pi\sqrt{m/k},\quad Z=\frac{zk}{mg},\quad U=u/\frac{g}{2\pi}\sqrt{\frac{m}{k}},\quad V=v/\frac{g}{2\pi}\sqrt{\frac{m}{k}}$$

În aceste condiţii ecuaţiile sistemului devin:

$$\frac{dZ}{dT}=V$$

$$\frac{dV}{dT}=-(2\pi)^2Z+\beta c_l U\sqrt{U^2+V^2}$$

Unde: $\beta=\frac{\rho fgS}{2k}$.
Vom scrie un program Matlab în care vom lua: $\rho_f = 1.22kg/m^3$, $m=3kg$, $k=980kg/s^2$, $g = 9.81m/s^2$, $S=0.3\ m^2$, $\alpha_0=1$.

Exemplu .
Coduri Matlab

```
function yprim = aripv(x,y)
global U;
yprim=zeros(2,1);
UU=U*(1+0.1*sin(2*x));
V=y(2)+0.14*U*sin(0.5*x);
alpha0=10*pi/180;
beta=0.00183;
alpha=alpha0-atan(V/UU);
cl=0;
if abs(alpha)<=pi/10
    cl=2*pi*alpha;
end;
yprim(1)=y(2);
yprim(2)=-4*pi^2*y(1)+beta*cl*UU*sqrt(UU^2+V^2);
end
```

Câmpul vectorial:

```
[t,x]=ode45('aripv',[0,30],[0,0]);
plot(t,x(:,1));
title('Vibratiile unei aripi de avion');
grid
xlabel('T');
ylabel('Z');
```

Apelul din linia de comanda:

```
>>global U;U=100;aripvp;
```

Rezultă graficul:



10.5.3. Modelul descompunerii termice a ozonului

În lucrarea **Lapidus L.** ***The occurence and numerical solution of physical and chemical systems widely varyng time pp.187-200 in Stiff Differential Systems editura Penum, New-York 1973*** propune următorul sistem de ecuaţii diferenţiale care simulează acest fenomen:

$$\frac{dx}{dt} = -x - xy + \varepsilon ky$$

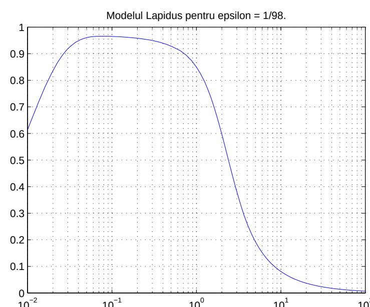
$$\varepsilon\frac{dy}{dt} = x - xy - \varepsilon ky$$

Aici variabila x reprezintă concentraţia de ozon, variabila y concentraţia de oxigen, iar pentru parametrii reali `ε = 1/98 şi k=3.`
Rezolvăm aceast sistem de ecuaţii diferenţiale folosind valorile iniţiale : x(0) = 1, y(0) = 0 iar intervalul de integrare este [0,240].

Exemplul 3.
Codul Matlab

```
function lapidus
global epsilon kappa
epsilon = 1/98;
kappa = 3;
options = odeset('Stats','on');
[t,y] = ode15s(@f,[0,240],[1;0],options);
semilogx(t,y(:,2))
grid on
axis([0.01 100 0 1])
title('Modelul Lapidus pentru epsilon = 1/98.');
%===========================================================
function dydt = f(t,y)
global epsilon kappa
dydt = [0; 0];
dydt(1) = - y(1) - y(1)*y(2) + epsilon*kappa*y(2);
dydt(2) =  (y(1) - y(1)*y(2) - epsilon*kappa*y(2))/epsilon;
```

Sugerăm folosirea rezolvitorului `ode15s` datorită faptului că y′(t) este multiplicat cu ε.
În urma executării acestui program rezultă graficul:



10.6. Rezolvarea numerică a problemelor cu valori pe frontieră

10.6.1. Metode numerice

Metoda cu diferenţe

Fie problema cu valori pe frontieră:

$$\frac{d^2y}{dx^2} + A(x)\frac{dy}{dx} + B(x)y = C(x),\ x \in [a, b] \qquad \text{(BVP)}$$

$$y(a) = \alpha,\ y(b) = \beta$$

Pentru a rezolva acest tip de probleme în prima fază ele se discretizează de exemplu prin diferenţe finite.
Se consideră grila:

$$a = x_0, \ldots, x_i = ih, \ldots ; x_{n+1} = b, i = 0, 1, \ldots, n+1.$$

Unde: $h = \frac{1}{n+1}$.
Valorile lui y pe aceste puncte vor fi notate prin y_i, iar din formula lui ***Taylor*** avem pentru **h** mic:

$$y_{i+m} = y_i + mhy_i' + \frac{(mh)^2}{2!}\ y_i'' + \ldots$$

Rezultă:

$$y_{i-1} = y_i - hy_i' + \frac{(h)^2}{2}\ y_i'' - \frac{h^3}{6}\ y_i''' + \ldots$$

$$y_{i+1} = y_i + hy_i' + \frac{(h)^2}{2}\ y_i'' + \frac{h^3}{6}\ y_i''' + \ldots$$

Scăzând aceste relaţii obţinem *formula cu diferenţe finite progressive:*

$$y_i' = \frac{y_{i+1} - y_{i-1}}{h} - \frac{h}{2}\ y_i'' + \ldots$$

Procedând analog se obţine *formula cu diferenţe finite regresive:*

$$y_i' = \frac{y_{i+1} - y_{i-1}}{h} + \frac{h}{2}\ y_i'' + \ldots$$

şi cea cu *diferenţe centrate:*

$$y_i' = \frac{y_{i+1} - y_{i-1}}{2h} - \frac{h^2}{6}\ y_i''' + \ldots$$

Erorile de aproximare sunt O (h), pentru primele două metode şi O (h^2) pentru ultima. Derivata de ordin II se aproximează prin:

$$y_i'' = (y_{i+1} - 2y_i + y_{i-1})/h^2 + h^2/12\ y_i''' + \ldots$$

Înlocuind aceste formule în ecuaţia diferenţială şi ordonând termenii avem:

$$(1-\frac{h}{2}A_i)y_{i-1}+(h^2B_i-2)y_i+(1+\frac{h}{2}A_i)y_{i+1}=h^2C_i \quad i=1,...n.$$

Ceea ce reprezintă condiţia de verificare a ecuaţiei diferenţiale în nodurile interioare, unde prin A_i, B_i, C_i se înţeleg valorile lor în x_i.
În ceea ce priveşte condiţia la limită avem:

$$y_0=y_m \ , y_{n+1}=y_M$$

valori cunoscute care trec deci în membrul drept. Rezolvând acest sistem cu tehnicile de matrici rare, se obţin valorile aproximative ale soluţiei y în nodurile interioare ale grilei (matricea sistemului este diagonal dominantă, deci inversabilă).
Similar se procedează şi la sisteme de ecuaţii diferenţiale.

Metode de colocaţie

Aceste metode fac parte din metodele numerice de aproximare analitice. De obicei acestea au forma unei dezvoltări într-o serie trunchiată, fie după puterile lui x, fie în polinoame otogonale ***Cebâşev***, ***Laguerre etc.*** fie într-un alt sistem de funcţii de bază (de exemplu ***B-splines).***
Fie intervalul [a,b] şi o diviziune uniformă a lui , cu pasul $h=\frac{b-a}{N}$, N arbitrar şi în fiecare subinterval $[x_i, x_{i+1}]$ inserăm k puncte ce reprezintă rădăcinile unor polinoame ortogonale : Cebâşev, Legendre, etc , astfel se obţin n=N*k+2 **puncte de colocaţie ale intervalului (se adaugă şi capetele intervalului).**

Metodele de colocaţie pe care le vom folosi în continuare constau în a determina soluţii aproximative $u_\Delta(x)$ ale soluţiei exacte y(x) ale problemei bilocale valabile pentru orice $x\in[a,b]$ sub forma:

$$u_\Delta(x)=\sum_{j=1}^{n} c_j\,\varphi_j(x) \ ,$$

unde $\varphi_j(x) \in P_{k,\Delta,n}$ unde $P_{k,\Delta,n}$ este mulţimea polinoamelor de grad cel mult k definite pe diviziunea Δ, a intervalului [a,b] , $\varphi_j(x)$ j=0,1,2,...n sunt funcţii liniar independente, iar c_j sunt parametrii reali. Aceşti parametrii sunt determinaţi, astfel încât $u_\Delta(x)$ să satisfacă ecuaţia diferenţială pe punctele de colocaţie a intervalului [a,b] şi în plus să verifice condiţiile la limită.
Detalii privind existenţa şi unicitatea acestor soluţii se pot regăsii în lucrarea [22].
Observaţie
Malabul pune la dispoziţia utilizatorilor un rezolvitor eficient `bvp4c`,îmbunătăţit prin `bvp5c` datorat lui Carl de Boor şi bazat pe funcţii B-splines.

Controlul erorii în cazul aplicării rezolvitorului *`bvp4c`*
Se face evaluând:

$$|u_\Delta(x_i)-y_{\Delta,i}| \le \tau|u_\Delta(x_i)| + \tau$$

Unde prin $y_{\Delta,i}$ am notat valorile soluţiei exacte a problemei BVP pe punctele de colocaţie, iar prin τ toleranţa.
Evident dacă $|u_\Delta(x_i)| \le 1$ controlul devine:

$$|u_\Delta(x_i)-y_{\Delta,i}| \le \tau \ ,$$

iar dacă $|u_\Delta(x_i)| > 1$ atunci:

$$|u_\Delta(x_i)-y_{\Delta,i}| \le \tau|u_\Delta(x_i)|$$

10.7.2. Forma liniară

În continuare vom trata problema BVP cu condiţii în interiorul intervalului, considerând că problema clasică se poate obţine pentru a=c, b=d.
Acest lucru este motivat de faptul că dacă intervalul de integrare este infinit, sau condiţiile la limită sunt infinite apar foarte multe probleme de aceea vom trata în continuare şi problemele bilocale de următoarea formă:

$$y''(x)+q(x)y(x)=r(x), \qquad a\le x\le b$$

cu condiţii în interiorul intervalului de integrare:

$$y(c)=\gamma,\ \ y(d)=\delta;\ \ a<c<d<b$$

În continuare vom considera că sunt îndeplinite condiţiile de existenţă şi unicitate a soluţiilor acestor probleme.
În lucrarea [22] am rezolvat aceste probleme cu ***metoda colocaţiei cu funcţii B-spline şi polinoame ortogonale de tip Cebâşev*** .Vom da câteva exemple sugestive în acest sens , mai multe exemple se pot regăsi în lucrarea mai sus menţionată.

Exemplu 1
Problema lui Burden [2 pag.48]**,** care apare în calculul structurii de rezistenţă a anumitor construcţii:

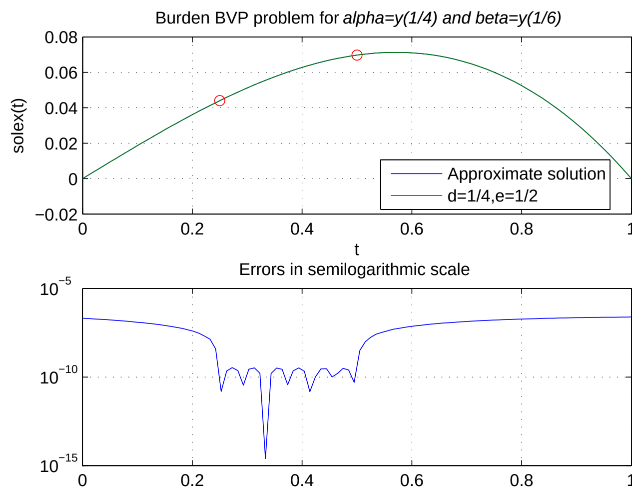
```
-y''-y = x,    x∈(0,1)
y((1/4))=((sin(1/4))/(sin1))-(1/4),
y((1/2))=((sin(1/2))/(sin1))-(1/2).
```

Codul Matlab este:

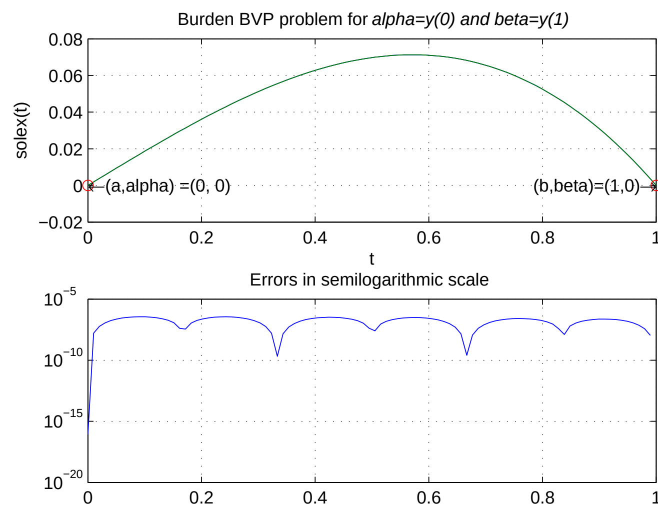
```
%numărul punctelor de colocaţie N=nk+2,k=gradul B-spline
%metoda de colocaţie cu funcţii B-spline şi Runge-Kutta.

q = @(x) -ones(size(x));
r = @(x) x;
% Solutia exacta
solex= @(x) -x+sin(x)/sin(1);
a=0; b=1;d=1/4;e=1/2;
alpha=solex(d);
beta=solex(e);
t=linspace(a,b,100);
%numărul punctelor de colocaţie
n=3;k=3;
% Apelul funcţiei tic pentru contorizarea timpului de execuţie
tic;
[x,y]=BVPcollocRK(q,r,a,b,d,e,alpha,beta,n,k,t);
toc;
subplot(2,1,1)
% Soluţia
plot(t,solex(t),x,y,[d,e],[alpha,beta],'o')
xlabel('t')
ylabel('solex(t)')
title('Burden BVP problem for alpha=y(1/4) and beta =y(1/6)')
legend('Approximate solution','Exact solution','d=1/4,e=1/2',0)
% Erorile
subplot(2,1,2)
semilogy(x,abs(solex(x)-y),'-')
title('Errors in semilogarithmic scale ')
grid on
```

În urma execuţiei acestui program rezultă graficele:



Dacă a=d, e=b problema de vine o problemă clasică (BVP), rezultatele se pot vedea în graficul următor:



Un alt exemplu este cazul liniar în care soluţia este oscilantă, adică există mai mult de două zerouri în intervalul de integrare.

Exemplu 2

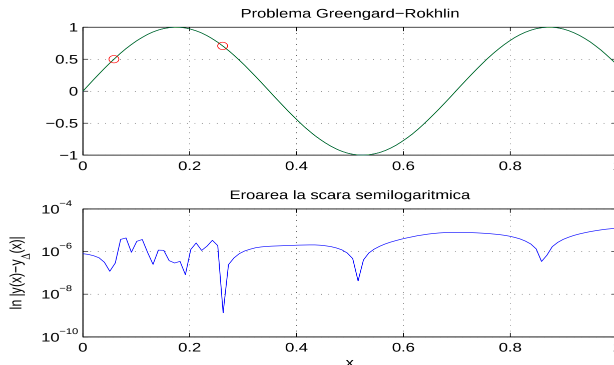
Fiind dată problema bilocală cunoscută în literatura de specialitate ca problema lui ***Greengard şi Rohlin sau ecuaţia cuardei vibrante într-un domeniu de frecvenţă [2, pp.68]:***

```
y''+86*y=5*sin(9*t);
y(pi/54)=1/2;y(pi/12)=sqrt(3)/2.
```

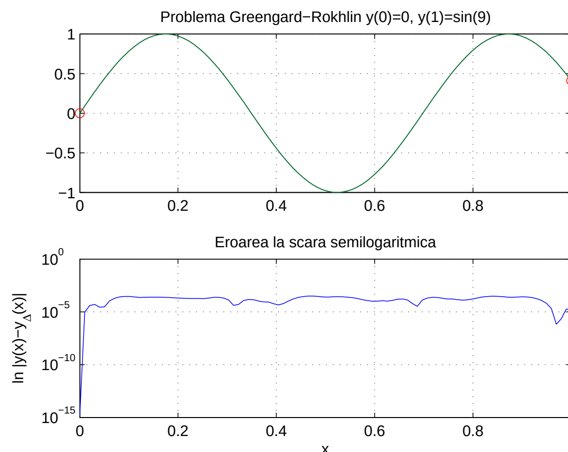
Codul Matlab este:

```
tic;
q = @(x) -86*ones(size(x));
r = @(x) -5*sin(9*x);
solex= @(x) sin(9*x);
a=0; b=1;
%condiţii  in interiorul intervalului
d=pi/54;
e=pi/12;
alpha=solex(d);
beta=solex(e);
t=linspace(a,b,100);
n=3;
k=3;
tic;
[x,y]=BVPcollocRK(q,r,a,b,d,e,alpha,beta,n,k,t);
toc;
subplot(2,1,1)
plot(t,solex(t),x,y,[d,e],[alpha,beta],'o')
grid on
title('Problema Greengard-Rokhlin')
subplot(2,1,2),semilogy(x,abs(solex(x)-y),'-')
xlabel('x','FontSize',10),
ylabel('ln |y(x)-y_{\Delta}(x)|','FontSize',10)
grid on
title('Eroarea la scara semilogaritmica')
```

În urma execuţiei acestui program Matlab obţinem rezultatul:



Analog dacă a=e, b=d obţinem rezultatul :



10.6.3. Forma neliniară

Formularea problemei:

Fie ecuaţia diferenţială **de ordin II** neliniară şi neomogenă cu condiţii în interiorul intervalului [a,b].

$$y''(x)=f(x,y(x)); \quad x \in [a,b]$$

cu condiţii la limită de tip Dirichlet neomogene date în interiorul intervalului de integrare.

$$y(c)=\alpha, \ y(d)=\beta, \ a \le c \le d \le b.$$

Unele din aceste tipuri de probleme nu au soluţii analitice, deci singura posibilitate de rezolvare a lor sunt metodele numerice. Amintim câteva dintre aceste metode (31) :

- metoda shutting,
- metoda lui Newton,
- metoda colocaţiei.

În această lucrare ne vom ocupa să prezentăm exemple de utilizare ale următoarelor metode de colocaţie :

- metoda globală cu **funcţii B-splines**,
- metoda combinată **polinoame ortogonale Cebâşev** şi **Runge-Kutta**,
- metoda combinată **B-splines şi Runge-Kutta.**

Metodele combinate presupunem descompunerea intervalului de integrare în [a,c], [c,d], [d,b], astfel :

- pe intervalele [a,c] şi [d,b] tratăm problema (IVP) cu rezolvitorul `ode45`,
- [c,d] folosim metode de colocaţie cu B-spline sau polinoame ortogonale **Cebâşev.** De asemenea suntem interesaţi de costurile de execuţie (run-times).

Exemplu 3

Următoarea problemă cunoscută în literatura de specialitate ca **Problema lui Bratu [22, pag. 42],** şi apare în modelul combustiei spontane este:

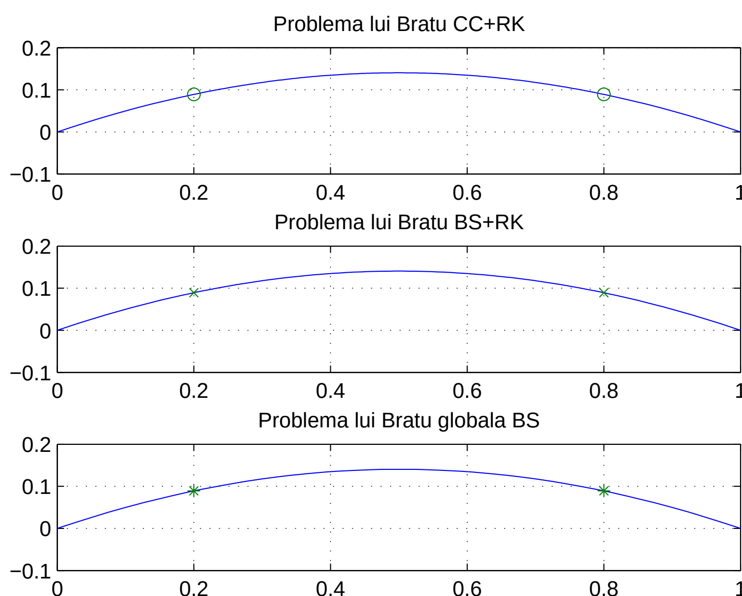
```
u''+exp(u)=0;
u(0.2)=u(0.8)=0.08918993462883;
folosim ca solutie de start:
y(x)=x(1-x)
```

Pentru calculul valorilor în `u(0.2)=u(0.8)=0.08918993462883;` am folosit cod Maple.

Codul Matlab este:

```
% Problema lui BRATU
%Folosim pentru  metodele C.C+R.K si B.s+R.K aceasi solutie de start:
%y(x)=x(1-x)
%Pentru metoda B.S. Globala solutia de start este:
%y(x)=3.9x(1-x)/7;
%u''+exp(u)=0; u(0.2)=u(0.8)=0.08918993462883;
f=@(x,y) -exp(y);
df=@(x,y) -exp(y);
err=1e-10;%criteriul de stop
NMAX=50;%numarul maxim de iteratii
N=15;%numarul punctelor de diviziune
f0=@(x) x.*(1-x);%solutia de start
a=0; b=1;
%condiţii din interirul intervalului (0,1)
c=0.2;
d=0.8;
%valorile alpha şi beta sunt calculate cu Maple
alpha=0.08918993462883; bet=0.08918993462883;
%metoda pseudospectrala+Runge-Kutta
tic;
[x,y,ni]=solvepolylocalceb(N,f,df,a,b,c,d,alpha,bet,f0,err,NMAX);
toc;
ni;%numar de iteratii
subplot(3,1,1);
plot(x,y,[c,d],[alpha,bet],'o')
grid on
title('Problema lui Bratu CC+RK')
%metoda B-splines+Runge-Kutta
k=3;d=0.2;
e=0.8;N=15;
t=linspace(a,b,100);
tic;
[x,y]=polycalnlinRK(f,df,a,b,d,e,alpha,bet,N,k,f0,t,err);
toc;
subplot(3,1,2);
plot(x,y,[d,e],[alpha,bet],'x')
grid on
title('Problema lui Bratu BS+RK')
%metoda globala B-spline
N=15;
startv=@(x) 3.9*x.*(1-x)/7;
[x,y]=polycollocnelin(f,df,a,b,d,e,alpha,bet,N,k,startv,t,err);
toc;
subplot(3,1,3);
plot(x,y,[d,e],[alpha,bet],'*')
grid on
title('Problema lui Bratu globala BS')
```

În urma execuţiei scriptului de mai sus rezultă graficul:



Am folosit pentru compararea costurilor funcţiile Matlab `tic-toc`, iar rezultatele se pot vedea în tabelul următor:

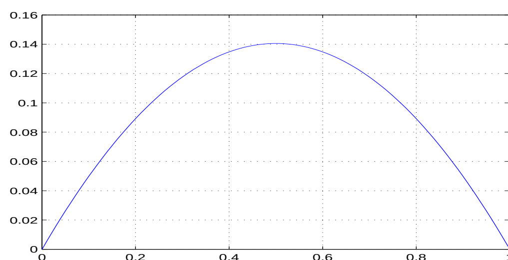
ε	C.C + R.K	B.S +R.K	Global B.S
10^{-5}	0.0540	0.035	0.021
10^{-6}	0.077	0.043	0.023
10^{-7}	0.049	0.025	0.024
10^{-8}	0.055	0.031	0.031
10^{-9}	0.054	0.036	0.030
10^{-10}	0.058	0.028	0.026

Unde prin ε am notat eroarea.

Problema clasică cu valori pe frontieră:

```
u''+exp(u)=0;
u(0)=u(1)=0;
```

Folosind codul Matlab din lucrarea [8,pag:168-169] care foloseste rezolvitorul `BVP4c` se obţine graficul:



Costurile de execuţie furnizate de funcţiile MATLAB `tic şi toc` sunt: 0.629355 seconds.

Concluzie
Costurile mai mari se datorează faptului că pentru a obţine soluţia aproximativă autorii aplică de mai multe iteraţii rezolvitorului `BVP4c` şi extrapolează de fiecare dată rezultatele intermediare cu `BVPINIT`.

10.6.4. Condiţii la limită infinite

Acest tip de probleme a început să fie studiat în ultimii 50 ani, deoarece ele apar în modele matematice din medicină, chimie, biologie, reţele Neuronale , reactoare nucleare, compresia şi criptarea datelor etc.
Pentru a se înţelege mai bine acest tip de probleme vom da câteva exemple:

Exemplul 1 [8, pag. 155]
Vom trata în continuare ***modelul lui Thomas-Fermi*** care apare în procesul de dezintegrare a atomului de uraniu în reactoarele nucleare:

$$y''=x^{-1/2}y^{3/2}$$

cu condiţii pe frontieră:

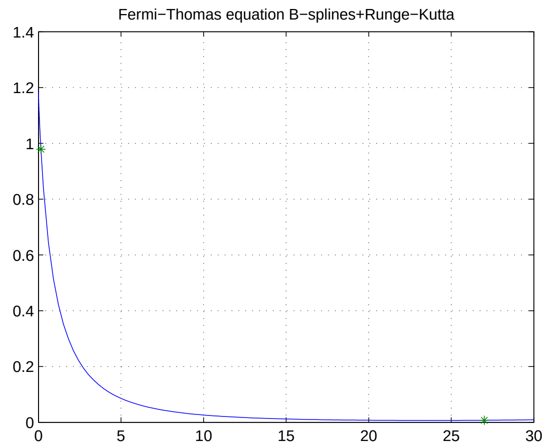
$$y(0)=1,\ y(\infty)=0$$

Se observă că x = 0 este un punct singular, mai mult $y(\infty) = 0$, deci abordarea unei astfel de problemă este foarte dificilă. ***Davis (1962)*** reuşeşte folosind o dezvoltare în serii de puteri să determine o soluţie raţională aproximativă :
$y(x)\cong 144x^{-3}$, $y'(x) \cong -432x^{-4}$.
Vom încerca o rezolvare numerică folosind o metodă combinată adică pe [0.15, 27] o metodă de colocaţie bazată colocaţie **cu B-spline**, iar pe [0, 0.15] şi [27,∞) **o metodă Runge-Kutta-Fehlberg (ode45).**

Codul Matlab
```
% Modelul Thomas Fermi: y''=x^(-1/2)*y^(3/2)
%Conditii la limită y(0)=1;y(infinity)=0;
%BS+RK method
f=@(x,y) x.^(-1/2).*y.^(3/2);
df=@(x,y) 3/2*x.^(-1/2).*y.^(1/2);
err=1e-10;
NMAX=5;
f0=@(x) 144*x.^(-3);
infty=10^25;
%interval
a=0.0001; b=30;
%number of collocation points
k=3;N=128;
%inner points
d=0.15;e=27;
 alpha=0.978608879909728;
 bet= 0.0073;
t=linspace(a,b,100);
tic;
profile on
[x,y]=polycalnlinRK(f,df,a,b,d,e,alpha,bet,N,k,f0,t,err);
toc;
profile viewer
profsave(profile('info'),'profile_results');
figure(3);
plot(x,y,[d,e],[alpha,bet],'*')
```

```
axis=[0 30 0 1];
grid on
title('modelul Fermi-Thomas B-splines+Runge-Kutta')
```

În urma execuţiei codului **Matlab** de mai sus se obţine graficul:

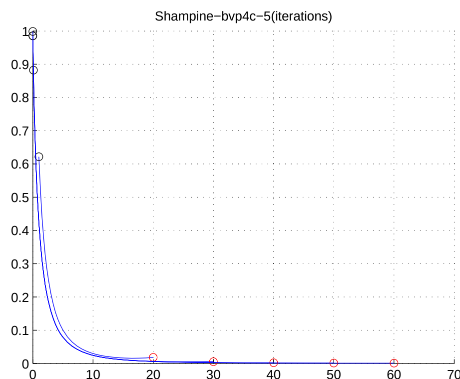


Costurile de execuţie sunt: 1.077779 seconds.
Aceaşi problemă tratată după ideea lui ***Gladwell*** [8 p. 155:156] folosind codul Matlab:

```
function sol = FermiShampine
%Rezolvare Bvp4c ideea propusă de Shampine
global d D
left = [1 0.1 0.01 0.01 0.001];
right = [20 30 40 50 60];
hold on
profile on
for i = 1:5
    tic
    d = left(i);
    D = right(i);
    if i == 1
        solinit = bvpinit(linspace(d,D,5),@guess,-1.6);
    else
        solinit = bvpinit(sol,[d,D]);
    end
    sol = bvp4c(@odes,@bcs,solinit);
    tt(i)=toc;
    figure(1)
    plot(sol.x,sol.y(1,:),sol.x(end),sol.y(1,end),'ro',...
         sol.x(1),sol.y(1,1),'ko');
     title('Shampine-bvp4c-5(iterations)')
    axis([0 70 0 1]);
       grid on
     drawnow
end
profile viewer
hold off
figure (2)
x = [0 sol.x];
y = [1 sol.y(1,:)];
```

```
plot(x,y)
title('Shampine-bvp4c-final')
axis([0 70 0 1])
grid on
sum(tt)
%==========================================================
function v = guess(x)
if x <= 1
    v = [1; 0];
else
    v = [ 144/x^3; -432/x^4 ];
end
function dydx = odes(x,y,p)
dydx = [ y(2); max(y(1),0)^(3/2) / sqrt(x) ];
function res = bcs(ya,yb,p)
global d D
res = zeros(3,1);
res(1:2) = ya - series(d,p);
res(3) =   yb(1) - 144/D^3 ;
function y = series(x,p)
yx = 1 + p*x + (4/3)*x^(3/2) + (2/5)*p*x^(5/2);
ypx = p + 2*x^(1/2) + p*x^(3/2);
y = [yx; ypx];
```

Se obţine rezultatul:



Costurile sunt 1.7644 seconds (mult mai mari deoarece se fac cinci iteraţii pentru a obţine soluţia aproximativă).

Exemplul 2 [8, p. 153]
Un al doilea **exemplu** este problema cunoscută in literatura de specialitate ca **modelul** **`Ames & Lohner`** şi apare în fenomenul de poluare a apelor curgătoare:

```
y''-100y+10y^2=0; in [0,infint]
      y(0)=1, y(infinit)=0;
```

Autorii determină o soluţie aproximativă exponenţială, obţinută eliminând din ecuaţia diferenţială termenul `10y^2`, deci:

$$y(x)=e^{-10x}$$

Vom descompune problema astfel:

- O zonă tranzitorie [0,a] în care vom folosii o **metodă de colocaţie cu funcţii B-spline,**
- O zonă exponenţială [a,40] în care vom folosi **o metoda Runge-Kutta-Fehlberg.**

În alegerea punctului `x = a < 40`, trebuie avut în vedere zona de stabilitate a metodei **B-spline.**

Codul Matlab este:

```
clear all
close all
f=@(t,y) 100*y-10*y.^2;
fd=@(t,y) -20*y+100;
fe=@(x) exp(-10*x);%soluţia exactă
startv=@(t) 1./(1+t).^3;
NMAX=5;%Numărul maxim de iteraţii
%Datele iniţiale
a=0; b=40;
d=a; e=10;% puncte interioare
g=0.1;% punct de control
alpha=fe(a);
beta=0;
 gama= fe(e);
 delta=fe(g);
% număr grade de libertate  p=N*k, unde  k este gradul funcţiei B-
spline
N=100;
k=3;
t=linspace(a,b,100);
err=1e-15;
tic;
[x,y]=polycalnlinRK(f,fd,a,b,d,e,alpha,beta,N,...
    k,startv,t,err);
toc;
%Plotăm  soluţia de start şi soluţia aproximativă
Subplot(2,1,1)
plot(t,startv(t),'--',x,y,'-
r',[a,g,e,b],[alpha,delta,gama,beta],'*k');
xlabel('x','FontSize',10); grid;
title('B-splines+Runge-Kutta')
axis([0.0001 40 0.0001 1]);
legend('Start solution',' Aprox solution',' Inner points')
text(a,alpha,'\leftarrow(a,alpha)',...
     ...,
     'FontSize',7)
 text(g,delta,'\leftarrow (g,delta)',...
     ...,
     'FontSize',7)
text(e,gama,'\rightarrow  (e,gama)',...
     'HorizontalAlignment','right',...
     'FontSize',7)
 text(b,beta,'\rightarrow(b,beta)',...
     'HorizontalAlignment','right',...
     'FontSize',7)
hold on
% Eroarea
subplot(2,1,2)
axis([0.00001 40 0.00001 1]);
```

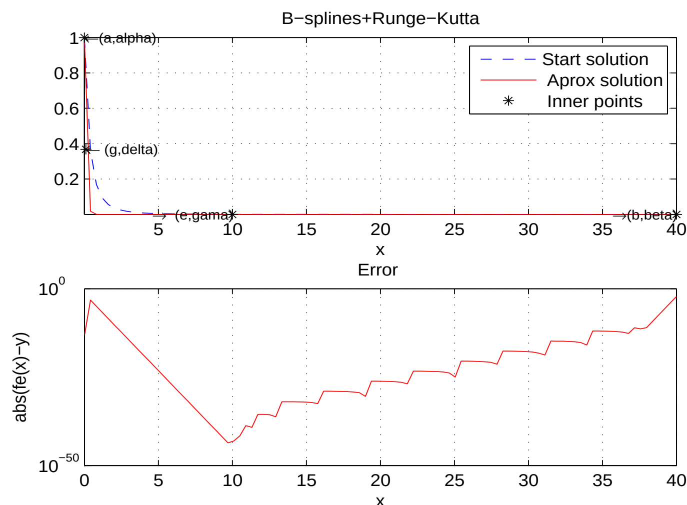
```
semilogy(x,abs(fe(x)-y),'-r');grid;
xlabel('x','FontSize',10)
ylabel('abs(fe(x)-y)','FontSize',10)
grid on
title('Error ')
```

Funcţia `polycalnlinRK` este:

```
function
[x,y]=polycalnlinRK(f,fd,a,b,d,e,alpha,beta,N,k,startv,t,err)

%N - # numărul de subintervale
%k - # numărul punctelor de colocaţie
%startv - functia de start
%err - eroarea
ti=t(t>=d & t<=e);
tright=t(t>e);
sp=BVPcollocnelin(f,fd,d,e,alpha,beta,N,k,startv,err);
yi=fnval(sp,ti);
ydi=fnval(fnder(sp),[d,e]);
x=ti';
y=yi';
opts=odeset('AbsTol',1e-12,'Reltol',1e-11,'Stats','on');
 if ~isempty(tright)
    tright=[e,sort(tright)];
    [tr,wr]=ode45(@rhs,tright,[beta,ydi(2)],opts);
    x=[x;tr(2:end)];
    y=[y;wr(2:end,1)];
    fprintf('ydi(2)''(0) is about
%6.4f.\n',sol.parameters,'Stats','on');
end
    function dy=rhs(x,u)
        dy=[u(2); f(x,u(1))];
    end
end
```

În urma executării programului rezultă graficele:

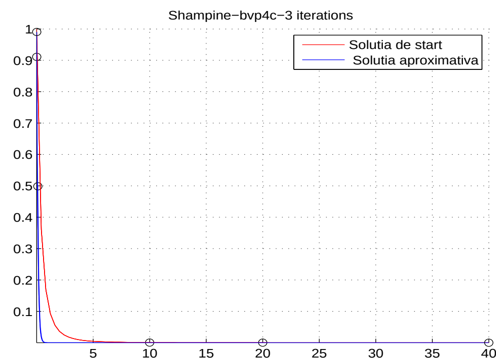


Costurile de execuţie sunt : 1.047822 seconds.
Aceaşi problemă poate fi tratată conform lucrarii [8 p. 153] cu ajutorul codului Matlab:

```
function sol = AmesShampine1
clear all
close all
global d D
fe=@(x) exp(-10*x);
startvv=@(x) 1./(1+x).^3;
left = [0.1   0.01 0.001];
right = [10    20    40];
hold on
%profile on
for i = 1:3
    tic
    d = left(i);
    D = right(i);
    if i == 1
        solinit = bvpinit(linspace(d,D,5),@startv,-1.6);
    else
        solinit = bvpinit(sol,[d,D]);
    end
    sol = bvp4c(@odes,@bcs,solinit);
    tt(i)=toc;
    t=linspace(0,40,100);
    figure(1)
plot(t,startvv(t),'r-
soll.x,sol.y(1,:),sol.x(end),sol.y(1,end),'ko',...
         sol.x(1),sol.y(1,1),'ko')
     legend('Solutia de start',' Solutia aproximativa')
     title('Shampine-bvp4c-3 iterations ')
    axis([0.0001 40 0.0001 1]);
       grid on
    drawnow
end
x = [0 sol.x];
y = [1 sol.y(1,:)];
figure(2)
axis([0.0001 40 0.0001 1]);
semilogy(x,abs(fe(x)-y),'-m');grid;
xlabel('x','FontSize',10)
ylabel('abs(fe(x)-y)','FontSize',10)
grid on
title('Error ')
sum(tt)
%============================================================
function v = startv(x)
if x <= 1
    v = [1; 0];
else
    v = [ 1/(1+x)^3; -3/(1+x)^4 ];
end
function dydx = odes(x,y,p)
dydx = [ y(2); 10*max(y(1),0)*(10-max(y(1),0));];
function res = bcs(ya,yb,p)
global d D
res = zeros(3,1);
res(1:2) = ya - series(d,p);
```

```
res(3) =   yb(1) - 1/(D+1)^3 ;
function y = series(x,p)
yx = 1 + p*x + (4/3)*x^(3/2) + (2/5)*p*x^(5/2);
ypx = p + 2*x^(1/2) + p*x^(3/2);
y = [yx; ypx];
```

După trei iteraţii se obţine rezultatul:



Costurile sunt: 6.1895 seconds acest lucru se datoreşte faptului că sunt aplicate trei iteraţii pentru a se obţine soluţia aproximativă.

Exemplul 3 [18]

Se dă problema bilocală:

`y''+2*pi^2*exp(-y)=0; in [0,1]`

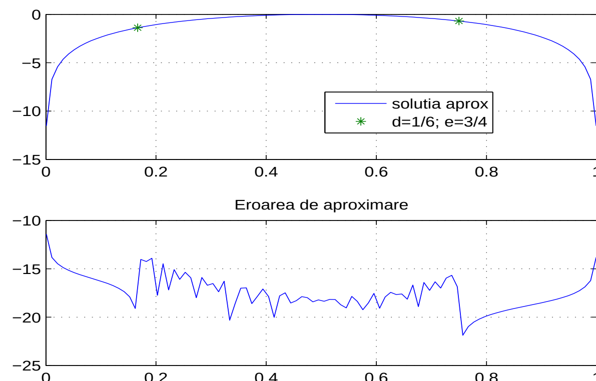
Soluţia acestei probleme este :

`y=ln(sin(pi*x)^2)`

Observăm că: `y(0)= -∞, y(1)=-∞`, de aceea nu o vom putea trata cu rezolvitorul `bvp4c`, ci luând drept condiţii în interiorul intervalului [0,1] astfel:

`y(1/6)=-2*ln(2), y(1/4)=-ln(2)`

Folosind **codul Matlab** din lucrarea [22 pag 50:60] vom obţine următorul grafic:



Exemplu 4 [8, p. 189:190]

Vom da un alt exemplu cunoscut în literatura de specialitate ca **problema lui Fischer, Kolmogorov, Petrovski, Piscunov (1937) (FKPP) problemă care apare în fenomenul de propagare a flăcării în reactoarele nucleare.**

Este cea mai simplă ecuaţie de reacţie-difuzie provenită din ecuaţia căldurii prin adăugarea unei funcţii neliniare f(u):

$$u_t = \Delta u + f(u)$$

Dacă: $u(x,t)=u(z)$, cu $z=x-ct$; unde c= viteza undei, atunci:

$$u''+cu'+u(1-u)=0.$$

Kolmogorov, Petrovski, Piscunov în lucrarea ***Study of the diffusion equation with growth of the quantity matter and its application to a biological problem Bull. State Univ. Moscow (1937)*** demonstrează că soluţia acestei ecuaţii depinde de data iniţială $u_0(z)$, funcţie monotonă şi continuă, $u_0(z)=1$, dacă $x < a$ şi $u_0(z)=0$, dacă $x > b$, unde $-\infty < a < b < \infty$. Mai mult ei arată că pentru $c \geq 2$ soluţia acestei probleme pentru un c fixat este unică şi $u(0) =1/2$.

În mod natural se impun condiţii la limită :

$$u(-\infty)=1, u(\infty)=0.$$

În lucrarea **Mathematical Biology ,** ***J.D. Murray în capitolul Biological Waves: Single – Specis Models p: 438 -445 Editura Berlin-Springer Verlag 1993*** , studiază această problemă în planul fazelor (u, u'). Evident punctele fixe sunt (0,0) şi (1,0).

Studiind stabilitatea acestui sistem dinamic el demonstrează că pentru $c \geq 2$ punctul critic (0,0) este ***nod stabil***, iar (1,0) este ***punct şa***. Examinând traiectoriile în planul fazelor, ***J.D. Murray*** justifică că pentru $c \geq 2$ există o soluţie unică astfel încât într-o vecinătate a punctului (0,0) :

$$u'(z) \sim \beta\, u(z).$$

unde $\beta=(-c+\sqrt{c^2-4})/2$, iar în vecinătatea puctului (1,0):

$$(u(z)-1)' \sim \alpha(u(z)-1).$$

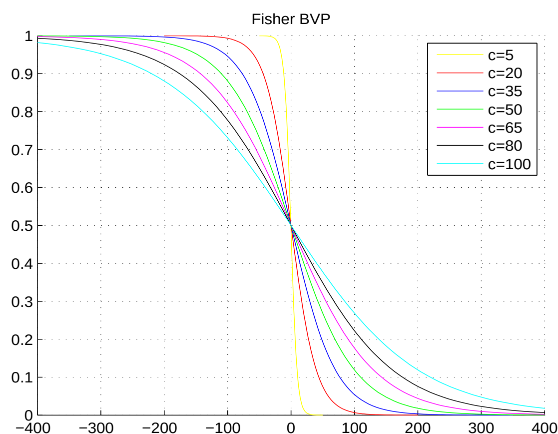
unde $\alpha=(-c+\sqrt{c^2+4})/2$.

Folosind aceste rezultate ***Shampine-Gladwell-Thomson*** în lucrarea [8, pp. 183-189] scriu următorul cod Matlab pentru următoarele valori ale parametrului

$c \in \{5,20,35,50,65,80,100\}$.

```
function Fisher
global c alpha beta infty
options=[];
%optiuni
options=bvpset(options,'Vectorized','on');
options=bvpset(options,'FJacobian',@odeJac);
options=bvpset(options,'BCJacobian',@bcJac);
color= ['y','r','b','g','m','k','c'];
ws=[5,20,35,50,65,80,100];
hold on
for i=1:7
    c=ws(i);
    alpha=(-c+sqrt(c^2+4))/2;
    beta=(-c+sqrt(c^2-4))/2;
    infty=10*c;
    solinit=bvpinit(linspace(-infty,infty,20),@guess);
    tic;
     sol=bvp4c(@ode,@bc,solinit,options);
    toc;
```

```
     figure(1)
    plot(sol.x,sol.y(1,:),color(i));
    title('Fisher BVP');
    axis([-400 400 0 1]);
    grid on
    hold on
    drawnow
end
legend('c=5','c=20','c=35','c=50','c=65','c=80','c=100',7);
hold off
function v = guess(z)
global c alpha beta infty
if z>0
    v=[exp(beta*z); beta*exp(beta*z)];
else
    v=[(1-exp(alpha*z)); -alpha*exp(alpha*z)];
end
function dydz = ode(z,y)
global c alpha beta infty
dydz = [y(2,:);-(c*y(2,:)+y(1,:).*(1-y(1,:)))];
% Funcţia care calculează Jacobianul
    function dFdy = odeJac(z,y)
        global c alpha beta infty
        dFdy = [0,1
             (-1+2*y(1)),-c];
        function res = bc(ya,yb)
            global c alpha beta infty
res = [ya(2)/(ya(1)-1)-alpha
    yb(1)/exp(beta*infty)-1];
       function [dBCdya,dBCdyb] = bcJac(ya,yb)
                 global c alpha beta infty
dBCdya = [-ya(2)/(ya(1)-1)^2,1/(ya(1)-1 0 0];
dBCdyb = [0 0 1/exp(beta*infty),0];
```
În urma execuţiei acestui program rezultă graficul:



Se observă că pentru diferite valori ale lui c toate curbele trec prin punctul (0 ; 1/2)

Exemplul 5

În articolul ***Numerical Solution of Boundary Value Problems for Ordinary Differential Equations, SIAM, 1995, pp: 336-337, Uri Ascher, Robert Mattheij, Robert Russell*** propun rezolvarea următoarei probleme (BVP):

```
u'' + (2/x)*y' + [u-(1+2/x^2)]*u = 0, x∈[0,∞)
```

Dacă folosim substituţia: u=v*x şi înlocuim [0,∞) prin [0, L] obţinem următoarea problemă (BVP):

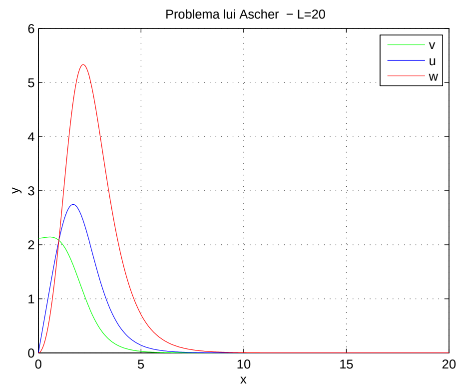
```
v'' + (4/x)*v' + (x*v-1)*v = 0, x∈[0,L]
v'(0)=0, v(L)+v'(L)=0
```

Folosind metoda sugerată de **Shampine, Gladwell şi Thomson** în lucrarea [8] scriem următorul program Matlab.

```
function Elmagbvp
close all, clear all
profile on
L=20;
S = [0  0
     0 -4];
options = bvpset('Singular-Term',S);
solinit = bvpinit(linspace(0,L,300),@ascherinit);
sol = bvp4c(@ascherode,@ascherbc,solinit,options);
profile viewer
plot(sol.x,sol.y(1,:),'g',sol.x,sol.x.*sol.y(1,:),'b',sol.x,sol.x.^2.
*sol.y(1,:),'r');
grid on
title('Problema lui Ascher','Color','g');
legend('v','u','w');
xlabel('x');
ylabel('y');
grid on
% -----------------------------------------------------------------
function dydx = ascherode(x,y)
dydx = [  y(2)
         -(x*y(1)-1)*y(1)];
end
% -----------------------------------------------------------------

function res = ascherbc(ya,yb)
res = [ ya(2)
        yb(1)+ yb(2) ];
end
% -----------------------------------------------------------------
function yinit = ascherinit(x)
% solutia de start
yinit = [  2*(x<=3/2)+2*exp(3/2-x).*(x>3/2);
          0*(x<=3/2)-2*exp(3/2-x).*(x>3/2) ];
end
%------------------------------------------------------------------
end
```

În urma execuţiei acestui program obţinem:



Timpul de execuţie stabilit folosind funcţiile ***Matlab*** este: 1.453751 seconds.

CAPITOLUL 11

REZOLVAREA ECUAŢIILOR CU DERIVATE PARŢIALE

11.1. Metode numerice pentru ecuaţii cu derivate parţiale

Marea parte a problemelor fizicii (şi nu numai) conduc la la ecuaţii cu derivate parţiale ale căror soluţii analitice sunt în general dificil de găsit şi utilizat . Metodele numerice reduc problema dată la diferite probleme standard, dar oferă numai rezultate discrete , adică valori în punctele unei grile sau coeficienţii unei serii trunchiate. Principalul aspect cantitativ avut în vedere este *acurateţea* unei metode date , adică capacitatea ei de a aproxima soluţia analitică atunci când instrumentele de aproximare devin suficient de fine. Alt aspect calitativ avut în vedere este *stabilitatea* metodei, adică capacitatea ei de a nu propaga şi acumula erorile din calculele anterioare.
Vom trce în revistă câteva din cele mai utilizate metode numerice , iar pentru detalii suplimentare care vizează aceste metode sugerăm lucrarea [21, pp. 192:214].

11.1.1. Modele continue şi discrete

Modelele de bază , care vor fi utilizate în exemplele prezentate în capitolul următor sunt:

- **Ecuaţia nestaţionară de advecţie-difuzie**, cazul bidimensional:

$$u_t+au_x+bu_y-v\Delta u=f$$

sau cazul unidimensional :

$$u_t+au_x-vu_{xx}=f$$

cu a,b coeficienţi constanţi sau variabili, cu condiţii iniţiale şi la limită asociate.

- **Ecuaţia staţionară de advecţie-difuzie,** cazul bidimensional

$$au_x+bu_y-v\Delta u=f$$

respectiv

$$au_x-vu_{xx}=f$$

cu condiţii la limită de tip ***Dirichlet sau Neumann.***
Ponderea termenilor convectivi faţă de cei difuzi se măsoară cu numărul lui ***Peclet:***

$$P_e=\frac{|a|L}{v}$$

unde L este lungimea domeniului de calculat.

- **Legi liniare de conservare**

$$u_t+\ (au)_x-(vu_x)_x=f$$

pentru cazul unidimensional şi

$$u_t+\ (au)_x+(bu)_y-v\nabla\cdot(\nabla u)=f$$

Vom reprezenta forma continuă a acestor modele prin:

$$Au=b$$

În practică nu se pot găsii soluţii analitice, deci se pune problema găsirii unei soluţii aproximative, care să descrie suficient de bine fenomenul fizic, acesta trebuie să fie un element u_h dintr-un spaţiu finit dimensional, calculabil printr-un efort finit dintr-un sistem finit dimensional de ecuaţii:

$$A_hu_h=b_h$$

11.1.2. Metoda cu diferenţe finite (MDF)

Metoda este bazată pe seriile ***Taylor*** şi descrie derivatele unei funcţii ca diferenţe dintre valorile acesteia pe diferite puncte. Cu alte cuvinte, înlocuieşte operatorul de derivare din A cu combinaţii de operatori de translaţie în A_h.
Cunoscând valorile funcţiei u şi a derivatelor acesteia în punctul x se pot aproxima valorile lui u în punctele vecine x+h sau x-h.

$$u(x+h)=u(x)+h\frac{du}{dx}+\frac{h^2}{2}\frac{d^2}{dx^2}u+\frac{h^3}{6}\frac{d^3}{dx^3}u+\cdots$$

$$u(x-h)=u(x)-h\frac{du}{dx}+\frac{h^2}{2}\frac{d^2}{dx^2}u-\frac{h^3}{6}\frac{d^3}{dx^3}u+\cdots$$

unde h este mic, iar derivatele lui u se calculează în x.
Folosind formulele de mai sus se pot obţine derivatele aproximative de ordin I sau II în x:

$$\frac{du}{dx}=\frac{1}{h}[u(x+h)-u(x)]+O(h) \quad \text{sau}$$

$$\frac{du}{dx}=\frac{1}{2h}[u(x+h)-u(x-h)]+O(h^2)$$

$$\frac{d^2}{dx^2}u=\frac{1}{h^2}[u(x-h)-2u(x)+u(x+h)]+O(h^2)$$

Unde $O(h)$ şi $O(h^2)$ reprezintă ordinul de eroare.
Introducând aceste formule în ecuaţia Au-f = 0 se obţine :

$$A\,u_h-f_h=\sum_{k=1}^{n}a_k u(x+kh)-\sum_{k=1}^{n}b_k f(x+kh)=0$$

Prin acest procedeu o ecuaţie cu derivate parţiale valabilă într-un număr infinit de puncte, se transformă într-un sistem cu un număr finit de ecuaţii, care descriu relaţiile dintre valorile funcţiei necunoscute pe un număr finit de puncte din domeniu.
Dacă u este soluţia exactă şi u_h cea numerică, Au_h-f_h se numeşte *reziduu:*
dacă $Au_h-f_h=O(h^p)$, iar când $h\to 0$ se numeşte *ordin de trunchiere.*
Dacă eroarea de trunchiere tinde la zero, când $h\to 0$ procedeul de discretizare se numeşte *consistent.*

11.1.3. Metoda elementului finit (MEF)

Ideea acestei metode este că domeniul pe care se defineşte ecuaţia cu derivate parţiale se descopune într-un număr finit de subdomenii numite *elemente.*
Pe fiecare element se presupune o variaţie de formă simplă a funcţiilor necunoscute, iar apoi rezultatele sunt asamblate pentru a descrierea soluţiei numerice pe întreg domeniul.
În cazul unidimensional să presupunem că pe elementul respectiv funcţia necunoscută u variază liniar. Această funcţie se poate exprima deci pe elementul respectiv cu ajutorul valorilor ei pe capetele acestui element numite *noduri* , dacă variaţia ei este pătratică se mai utilizează valoarea lui u pe încă un punct din element de exemplu în mijlocul lui.

Cu această reprezentare derivata lui u pe acel element este constantă, iar derivata a doua este nulă şi nu mai poartă informaţii despre u. Pentru a elimina această situaţie, ecuaţiile unde apare şi derivata a doua se transformă în ecuaţii în care apare numai derivata întâia. Tehnica este cunoscută sub numele de deducere a *formulării variaţionale* şi constă în înmulţirea ecuaţiei diferenţiale cu o funcţie test, integrarea ecuaţiei obţinute pe domeniul respectiv şi apoi aplicarea unei formule de integrare prin părţi a termenilor conţinând derivate de ordin superior, cu scopul reducerii ordinului de derivare.

11.1.4. Metoda volumelor finite (MVF)

În esenţă discretizarea se efectuează prin transformări legate de fizica fenomenului studiat, prin conservarea unor mărimi în timpul calculelor numerice. Pentru aceasta se utilizează formularea integrală a legilor de conservare.

Domeniul fizic este considerat divizat în celule. Între momentele de timp t_n şi t_{n+1} variaţia unei anumite mărimi fizice, de exemplu a masei într-o celulă C_j notată cu

$$masa_j = vol(C_j) * densitatea_j$$

este dată de suma fluxurilor *flux* $_{jk}$ dintre C_j şi celulele vecine C_k

$$masa_j^{(n+1)} = masa_j^{(n)} + \sum_{k \in V(j)} flux_{jk}$$

Conservarea **masei totale** este asigurată de condiţiile:

$$flux_{jk} = -flux_{jk}$$

11.1.5. Metode spectrale

Aproximarea funcţiilor necunoscute se face cu ajutorul unor serii trunchiate de funcţii ortogonale, serii ***Fourier*** pentru probleme cu condiţii de periodicitate sau **de polinoame Cebâşev sau Laguerre** pentru probleme neperiodice:

$$u_N(x) = \sum_{k=0}^{N} \hat{u}_k c_k$$

unde valorile $\hat{u}_k$ sunt necunoscute de determinat.

În cazul problemei:

$$Lu = f, \quad x \in (a,b)$$
$$u(a) = u(b) = 0$$

un mod de determinare a coeficienţilor necunoscuţi este de a cere verificarea ecuaţiei date pe un anumit sistem de noduri, la care se adaugă condiţiile la limită:

$$Lu_N(x_k) - f(x_k) = 0, \quad k = 1, 2, \ldots, N-1.$$
$$u_N(a) = u_N(b) = 0$$

obţinând astfel ***metoda colocaţiei.***

Din interpretarea expresiei $u_N(x)$ ca **polinom de interpolare Lagrange** pe noduri x_k,

$$u_N(x) = \sum_{k=0}^{N} u_N(x_k) L_k(x)$$

unde $L_k(x_j) = \delta_{kj}$ necunoscutele de determinat sunt de fapt valorile $u_N(x_k)$.

Metodele spectrale se bucură de mult interes datorită faptului că eroarea dintre soluţia exactă u şi cea aproximată u_N este de ordin $1/N^s$ adică:

$$|u - u_N| \leq \frac{C}{N^s}$$

Unde s este legat de numărul de derivate continue pe care le admite u(x).

11.2. Aplicaţii în dinamica fluidelor

Să considerăm problema:

$$\frac{\vartheta^2 u}{\vartheta x^2} + \frac{\vartheta^2 u}{\vartheta y^2} = f(x,y), \quad \text{în } \Omega$$

$$u|_{\partial\Omega} = 0$$

Prin discretizare se ajunge la un sistem mare de ecuaţii algebrice, liniare.

Dăm câteva exemple de mişcări care conduc la o asemenea problemă (**ecuaţia lui Poisson**).

11.2.1. Domeniu rectangular cu vorticitate internă

Considerăm o mişcare generată de o vorticitate distribuită în cuva dreptunghiulară [-3 ,3] X [-2, 2]. Vorticitatea este vectorul în direcţia `Oz` de modul `q`, definit de rotaţionalul vitezei:

$$\nabla \times V = q$$

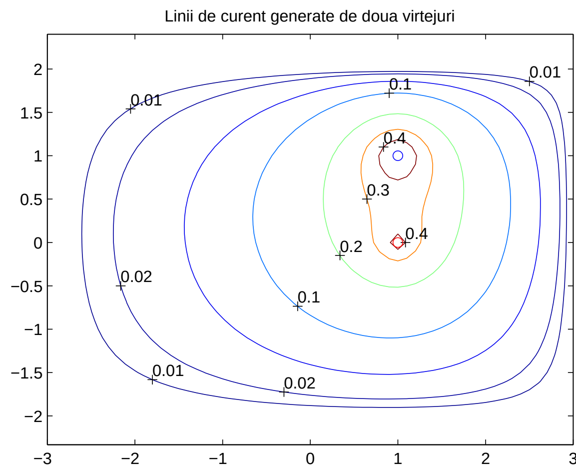
Utilizând relaţia dintre componentele vitezei şi funcţia de curent φ relaţia vectorială de mai sus se pune sub o formă scalară (trecând la modul):

$$\varphi_{xx} + \varphi_{yy} = -q(x,y)$$

Codul Matlab

```
plot(1,1,'bo',1,0,'ro');
hold off;
% programul este preluat din [21 pag 374]
x=-3:0.1:3;y=-2:0.1:2;
c=zeros(59,39); c(40,30)=-100;c(40,20)=-50;
e1=ones(59,1);
e2=ones(39,1);
D23=spdiags([e1 -2*e1 e1],[-1 0 1],59,59)*100;
D15=spdiags([e2 -2*e2 e2],[-1 0 1],39,39)*100;
[U,L]=eig(full(D23));
[V,P]=eig(full(D15'));
PSI=zeros(59,39);
C=inv(U)*c*V;
for i=1:59
    for j=1:39
        PSI(i,j)=C(i,j)/(L(i,i)+P(j,j));
    end
end
psi=U*PSI*inv(V);
S=zeros(61,41);
S(2:60,2:40)=psi;
cs=contour(x,y,S',[0.01 0.02 0.05 0.1 0.2 0.3 0.4]);
title('Linii de curent generate de doua virtejuri');
clabel(cs,'manual');axis('equal');
hold on;
```

În cazul unui domeniu [-3 3] × [-2 2] , cu vorticitatea generată de vârtejuri de intensitate 100 respectiv -50 plasate în punctele (1 1), respectiv (1 0) se obţin liniile de current din figura următoare



S-a utilizat procedeul exact de rezolvare a sistemului discret, iar funcţia de curent Ψ a fost aleasă să ia valoarea 0 pe linia de curent coincizând cu frontiera cuvei.

11.2.2. Curgerea fluidelor prin canale şi conducte

Să considerăm problema [21, pag 378] mişcării unui fluid staţionar, incompresibil, printr-o conductă rectilinie, de o secţiune constantă. Mişcarea este generată fie de un **gradient de presiune**, fie de forţa gravitaţională sau de mişcarea unor pereţi ai conductei în raport cu ceilalţi.

De asemenea, fluidul poate fi închis într-o conductă sau poate avea o suprafaţă liberă.

Să presupunem că conducta este infinit de lungă în direcţia axei Ox şi că de-alungul ei $\frac{\partial v}{\partial x} = 0$, aceasta înseamnă că fluidul are numai componenta nenulă a vitezei u în direcţia axială. Atunci ecuaţia continuităţii este satisfăcută automat, iar cea a momentului devine:

$$\frac{\vartheta^2 u}{\vartheta x^2} + \frac{\vartheta^2 u}{\vartheta y^2} = \frac{1}{\mu}\left(\frac{dp}{dx} - f_x\right)$$

Unde f_x este componenta forţei gravitaţionale pe unitate de volum în direcţia Ox. În absenţa unui gradient de presiune, singura forţă aplicată este $f_x = \rho\ g \sin\theta$, generată de acceleraţia gravitaţională.

Introducând mărimile adimensionalizate :

$$Y=y/L,\ Z=z/L,\ U = \frac{u\mu}{L*L\rho g \sin\theta}$$

Ecuaţia devine:

$$\frac{\vartheta^2 U}{\vartheta Y^2} + \frac{\vartheta^2 U}{\vartheta Z^2} = -1$$

Condiţiile la limită ataşate sunt cele de aderenţă la pereţii ficşi şi tensiunea de forfecare nulă pe suprafaţa liberă.

$$U|_{Z=0} = 0,\ U|_{Z=1} = 0$$
$$U|_{Y} = 0,\ \frac{\partial U}{\partial Y}|_{Y=1} = 0$$

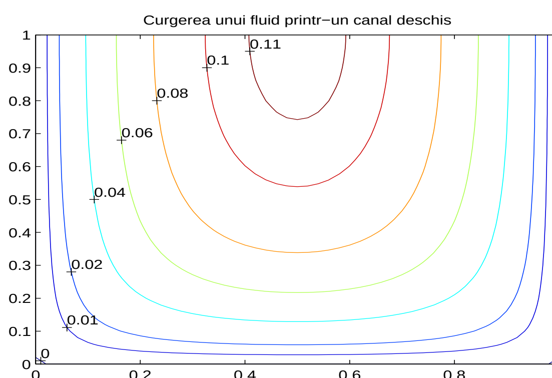
T. Petrila şi D. Trif în lucrarea [21] sugerează discretizarea cu diferenţe finite centrate de ordinul II pe o grilă de pas h care acoperă domeniul [0,1]×[0,1]. Având calculate vitezele în nodurile grilei se poate estima debitul de fluid trecând prin canal.

Codul Matlab este :

```
m=51;
n=m;
uv=zeros(m,n+1);un=uv;
h=1/(m-1);y=0:h:1;z=y;
alpha=cos(pi/m)+cos(pi/n);
omega=(8-4*sqrt(4-alpha^2))/alpha^2;
un(2:m-1,2:n)=0.04*ones(m-2,n-1);
iter=0;
while 1
    uv=un;
    for i=2:m-1
        for j=2:n
            if j==n
                un(i,j+1)= un(i,j-1);
            end;
            un(i,j)=(1-omega)*un(i,j)+omega/4*...
                (h^2+un(i-1,j)+un(i+1,j)+un(i,j-1)+un(i,j+1));
        end;
    end;
```

```
error=sum(sum(abs(un-uv)));
disp([iter error]);
%test daca s-a indeplinit conditia de eroare
if error<1.e-3
    break;
end;
iter=iter+1;
end
volrat= sum(sum(un(2:m-1,2:n-1))*h^2+...
     (sum(un(2:m-1,1)+un(2:m-1,n))+sum(un(1,2:n-1)'+...
    un(m,2:n-1)'))*h^2/2+(un(1,1)+un(1,n)+un(m,1)+un(m,n))*h^2/4);
cs=contour(y,z,un(1:m,1:m)',[eps,0.01,0.02,0.04,0.06,0.08,0.1,0.11]);
title('Curgerea unui fluid printr-un canal deschis');
clabel(cs,'manual');
% cu ajutorul comenzii clabel  se vor marca cu mouse-ul valorile
```

Rezultatele sunt prezentate în figura următoare:



11.2.3. Mişcarea unui fluid între două plăci plane

Mişcarea nestaţionară a unui fluid pus în mişcare de o placă care se deplasează în planul său, placă normală axei Oy, cu mişcare în direcţia axei Ox [21, p. 406].
Ecuaţia continuităţii se satisface imediat, iar ecuaţia ***Navier-Stokes*** se reduce la ecuaţia difuziei de forma:

$$\frac{\vartheta u}{\vartheta t} = \nu \frac{\vartheta^2 u}{\vartheta y^2}$$

Această ecuaţie este satisfăcută de vorticitatea :

$$\zeta = - \frac{\vartheta u}{\vartheta y}$$

a unei mişcări nestaţionare plane.
Dacă se trece la coordonate cilindrice, aceaşi situaţie apare la mişcări de fluide între cilindrii concentrici, iar găsirea câmpului vitezelor acestor mişcări se numeşte *problema Rayleigh generalizată.*

Exemplu

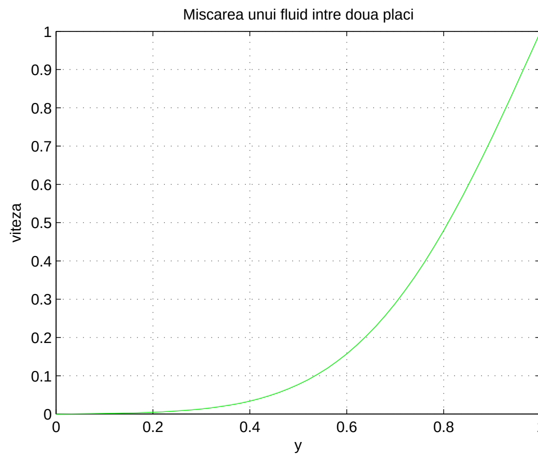
Fie un strat de apă conţinut între două plăci plane în repaus , distanţa de 1m. La t=0 placa de mai sus să o punem ţn mişcare cu viteza constantă $u_0 = 1$ m/s. Aceasta crează o vorticitate care difuzează în jos, conducând la redistribuirea corespunzătoare a vitezelor.

Pentru apă vom lua $\nu \approx 10^{-6}$ m^2/s, R = ¼, ceea ce conduce la un pas de timp $\tau = 625$s, iar pentru pasul h vom lua h = 0.05m.

Codul Matlab

```
h=1/50;r=0.25;u0=1;
y=0:h:1;
uv=zeros(51,1);
uv(51)=u0;
un=uv;
e=ones(49,1);
p=plot(y,uv,'EraseMode','None');
title('Miscarea unui fluid intre doua placi');
axis([0 1 0 1]);
grid
xlabel('y');ylabel('viteza');pause;
T=spdiags([r*e 1-2*r*e r*e],[-1 0 1],49,49);
b=zeros(49,1);b(49)=r*u0;
for i=1:400
    un(2:50)=T*uv(2:50)+b;
    set(p,'Color','k');
    drawnow;
    set(p,'Ydata',un,'Color','g');
    drawnow;
    uv=un;
end
```

Acest program ne oferă o animaţie descriind evoluţia în timp a profilului vitezei.



Exemplu

Vom rezolva problema determinării mişcării unui fluid într-un canal între două plăci plane, prin aplicarea bruscă a unui gradient de presiune $\frac{dp}{dx}$ în lungul acestuia. Ecuaţia mişcării este:

$$\frac{\vartheta u}{\vartheta t} = \nu \frac{\vartheta^2 u}{\vartheta y^2} - \frac{1}{p}\frac{dp}{dx}$$

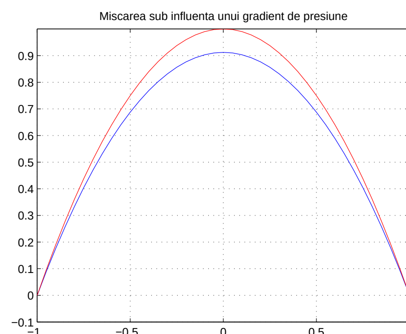
Dacă distanţa între plăci este 2L, condiţiile iniţiale şi la limită ataşate sunt:

$$u|_{t=0}, \quad -L \le y \le L, \quad u|_{y=\pm L}=0, \quad t > 0$$

Codul Matlab

```
r=0.4; h=0.05;
Y=(-1:h:1)';
wv=1-Y.^2;
e=ones(39,1);wn=zeros(41,1);
T=spdiags([r*e -(1+2*r)*e r*e],[-1 0 1],39,39);
p=plot(Y,1-Y.^2-wv,'EraseMode','none');
axis([-1 1 -0.1 1]);
hold on;
for i=1:1000
    wn(2:40)=T\(-wv(2:40));
    set(p,'Color','w');
    drawnow;
    set(p,'Ydata',1-Y.^2-wn,'Color','b');
    drawnow;
    wv=wn;
end
plot(Y,1-Y.^2,'r');
grid
title('Miscarea sub influenta unui gradient de presiune');
hold off;
```

Care realizează evoluţia în timp al profilului de viteză conform graficului următor:



Observaţie

Pentru domenii şi ecuaţii de o formă mai complicată se poate face apel la `PDE toolbox` `(bazate pe metoda elementului finit)` din `Matlab` prezentat în **capitolul 12.**

CAPITOLUL 12

REŢELE NEURONALE

12.1. Introducere

În acest capitol ne propunem să prezentăm ideile de bază ale calculului neural alături de principalele modele conexioniste ale inteligenţei artificiale şi am folosit lucrarea de referinţă: **M.Beale, H.Demuth** - ***Neural Network Toolbox. User Guide, 2008,***

În reţelele Neuronale informaţia nu mai este memorată în zone bine precizate, ca în cazul calculatoarelor standard, ci este memorată difuz în toată reţeaua. Memorarea se face stabilind valori corespunzătoare ale ponderilor conexiunilor sinaptice dintre neuronii reţelei.

Un alt element important, care este, probabil, principalul responsabil pentru succesul modelelor conexioniste, este capacitatea reţelelor neuronale de a învăţa din exemple.

În mod tradiţional, pentru a rezolva o problemă, trebuie să elaborăm un model (matematic, logic, lingvistic etc.) al acesteia. Apoi, pornind de la acest model, trebuie să indicăm o succesiune de operaţii reprezentând algoritmul de rezolvare a problemei. Există, însă, probleme practice de mare complexitate pentru care stabilirea unui algoritm, fie el şi unul aproximativ, este dificilă sau chiar imposibilă.

În acest caz, problema nu poate fi abordată folosind un calculator tradiţional, indiferent de resursele de memorie şi timp de calcul disponibil. Caracteristic reţelelor Neuronale este faptul că, pornind de la o mulţime de exemple, ele sunt capabile să sintetizeze în mod implicit un anumit model al problemei. Se poate spune că o reţea neurală construieşte singură algoritmul pentru rezolvarea unei probleme, dacă îi furnizăm o mulţime reprezentativă de cazuri particulare (exemple de instruire).

12.2. Procese de învăţare în sisteme cu inteligenţă artificială

Inteligenţa artificială, ca si in cazul inteligenţei biologice se dobândeşte printr-un proces continu si de durata de învăţare, de aceea problema învăţării ocupă un loc important în cercetarea maşinilor auto-instruibile (**machine learning**).

Prin învăţarea automata se înţelege studiul sistemelor capabile sa-şi îmbunătăţească performantele, utilizând o mulţime de date de instruire.

Sistemele cu inteligenţă artificiala obişnuite au capacităţi de învăţare foarte reduse sau nu au de loc. In cazul acestor sisteme cunoaşterea trebuie sa fie programată in interiorul lor. Dacă sistemele conţin o eroare , ele nu o vor putea corecta, indiferent de cate ori se executa procedura respectivă. Practic aceste sisteme nu-si pot îmbunătăţii performanţele prin experienţă si nici nu pot învăţa cunoştinţe specifice domeniului, prin experimentare. Aproape toate sistemele cu inteligentă artificială sunt sisteme deductive. Aceste sisteme pot trage concluzii din cunoaşterea încorporată sau furnizată, dar ele nu pot sa genereze singure noi cunoştinţe.

Pe măsura ce un sistem cu inteligenţa artificială are de rezolvat sarcini mai complexe, creşte şi cunoaşterea ce trebuie reprezentată în el (fapte, reguli, teorii). În general un sistem funcţionează bine, în concordanţă cu scopul fixat prin cunoaşterea furnizată, dar orice mişcare în afara competenţei sale face ca performanţele lui să scadă rapid. Acest fenomen este numit şi **fragilitatea cunoaşterii**.

Una din direcţiile de cercetare în privita maşinilor instruibile este **modelarea neurală** care dezvoltă sisteme instruibile pentru scopuri generale, care pornesc cu o cantitate mică de cunoştinţe iniţiale. Astfel de sisteme se numesc reţele neuronale sau sisteme cu autoorganizare sau sisteme conexioniste.

Un sistem de acest tip constă dintr-o reţea de elemente interconectate de **tip neuron**, care realizează anumite funcţii logice simple.

Un astfel de sistem învaţă prin modificarea intensităţii de conexiune dintre elemente, adică schimbând ponderile asociate acestor conexiuni. Cunoaşterea iniţială ce este furnizată sistemului este reprezentată de caracteristicile obiectelor considerate şi de o configuraţie iniţială a reţelei.

Sistemul învaţă construind o reprezentare simbolica a unei mulţimi date de concepte prin analiza conceptelor si contraexemplelor acestor concepte. Aceasta reprezentare poate fi sub forma de expresii logice, arbori de decizie, reguli de producţie sau reţele semantice.

Istoria Reţelelor Neuronale Artificiale **(RNA)** sau, simplu, a Reţelelor Neuronale începe cu modelul de neuron propus de către **W.McCulloch şi W.Pitts (un logician si un neurobiolog)** în lucrarea ***A logical calculus of the ideas immanent in nervous activity, Bull.Math.Biophys. pp.115-133, 1943*** şi este numit acest model neural, **neuronul MP.**

Modelul MP presupune că neuronul funcţionează ca un dispozitiv simplu, ale cărui intrări sunt ponderate. Ponderile pozitive sunt excitatoare, iar ponderile negative sunt inhibitoare. Dacă excitaţia totală, adică suma ponderată a intrărilor, depăşeşte un anumit prag, atunci neuronul este activat şi emite un semnal de ieşire (ieşirea are valoarea +1). Daca excitaţia totala este mai mica decât valoarea prag, atunci neuronul nu este activat şi ieşirea lui se consideră a fi zero.

Hebb (1949) a propus un mecanism calitativ ce descrie procesul prin care conexiunile sinaptice sunt modificate pentru a reflecta mecanismul de învăţare realizat de neuronii interconectaţi atunci când aceştia sunt influenţaţi de anumiţi stimuli ai mediului.

Rosenblatt (1959) a propus un dispozitiv numit perceptron. Perceptronul este bazat pe interconectarea unei mulţimi de neuroni artificiali şi reprezintă primul model de reţea neuronală artificială.

B. Widrow şi M.E. Hoff în lucrarea ***Adaptive switching circuits, IRE WESCON Convertion Record, New York : IRE Part 4,1960, 96-104,*** au propus un model neural numit ADALINE si o reţea cu elemente de acest tip numit MADALINE. ADALINE reprezintă acronimul ADAptive Linear Neuron sau ADAptive LINear Element. MADALINE este un acronim pentru Multiple-ADALINE.

Modelul ADALINE este în esenţa identic cu modelul perceptronului. Ieşirea este bipolară: +1 sau –1 şi este un dispozitiv adaptiv, în sensul că există o procedura bine definită de modificare a ponderilor pentru a permite dispozitivului să dea răspunsuri corecte pentru o intrare dată.

Reţelele Neuronale permit rezolvarea unor probleme complicate, pentru care nu avem un algoritm secvenţial, dar posedăm unele exemple de soluţii. Învăţând din aceste exemple (faza de instruire), reţeaua va fi capabila sa trateze cazuri similare (faza de lucru).

Calculatoarele obişnuite sunt, desigur, instrumente extrem de adecvate in rezolvarea unui spectru larg de probleme matematice, ştiinţifice, inginereşti. Calculatoarele işi dovedesc limitele in domenii in care omul excelează, cum ar fi percepţia si învăţarea din experienţă.

Într-un calculator obişnuit elementul esenţial este procesorul, caracterizat de viteza mare de lucru. În creier, elementele individuale de proces sunt celulele nervoase (neuronii). Ele sunt mai simple şi mai lente decât un procesor de calculator, însă sunt foarte numeroase. Conexiunile dintre neuroni sunt la fel de importante ca şi aceştia. Inteligenţa şi procesele memoriei rezidă în întreaga reţea de celule şi nu în neuronii individuali.

Cortexul cerebral este o reţea neurală naturală. O astfel de retea neurală are capacitatea de a gîndi, învăţa, simţi şi de a-şi aminti.

Reţelele Neuronale artificiale sunt reţele de modele de neuroni conectaţi prin intermediul unor sinapse ajustabile. Toate modelele de reţele neuronale se bazează pe interconectarea unor elemente simple de calcul dintr-o reţea densă de conexiuni.

Fiecare unitate de proces este capabilă să execute doar calcule simple, dar reţeaua, ca întreg, poate avea calităţi remarcabile in recunoaşterea formelor, rezolvarea problemelor pentru care nu posedăm un algoritm, învăţarea din exemple sau din experienţa. Paralelismul înalt şi capacitatea de învăţare reprezintă caracteristicile fundamentale ale reţelelor neuronale. Calculului neural implică două aspecte fundamentale: învăţarea şi reprezentarea cunoaşterii.

Reţelele Neuronale achiziţionează cunoaşterea prin instruire. O reţea neurală este instruită, dacă aplicarea unei mulţimi de vectori de intrare va produce ieşirile dorite. Cunoaşterea pe care reţeaua neuronala o dobândeşte este memorată de sinapsele neuronale, mai precis, în ponderile conexiunilor dintre neuroni.

Mulţi dintre algoritmi de instruire pot fi consideraţi ca avându-şi originea in modelul de învăţare propus de către **Donald Hebb (1949)** care propune un model al schimbărilor conexiunilor sinaptice dintre celulele nervoase. Conform modelului lui Hebb, intensitatea conexiunii sinaptice dintre doi neuroni (ponderea conexiunii) creşte de câte ori aceşti neuroni sunt activaţi simultan de un stimul al mediului. Acest mecanism este cunoscut de **regula lui Hebb de învăţare.**

Daca y_i este activarea neuronului i si exista o legătura sinaptică între neuroni i si j, atunci, in concordanţa cu legea lui Hebb, intensitatea conexiunii lor sinaptice este afectată de:

$$\Delta w_{ij} = c y_i y_j ,$$

unde c este un coeficient de proporţionalitate adecvat ce reprezintă constanta de instruire. Aceasta lege apare ca naturală in mulţi algoritmi de învăţare. În plus, există argumente neuro-biologice care sprijină ipoteza că stimulii mediului cauzează modificări sinaptice.

Acest mecanism este un model de învăţare nesupervizată în care drumurile neuronale des utilizate sunt intensificate (întărite). Acest model poate explica fenomenele de obişnuinţă si de învăţare prin repetare.

O reţea neurală artificială care foloseşte o învăţare hebbiană va determina o creştere a ponderilor reţelei cu o cantitate proporţională cu produsul nivelurilor de exercitare neuronale.

Fie $w_{ij}(n)$ ponderea conexiunii de la neuronul i la neuronul j înainte de ajustare şi $w_{ij}(n+1)$ ponderea acestei conexiuni după ajustare. Legea Hebb de învăţare se va scrie in acest caz sub forma:

$$w_{ij}(n+1) = w_{ij}(n) + c\ y_i y_j ,$$

unde y_i este iesirea neuronului i (intrarea neuronului j), iar y_j este iesirea neuronului j.

O variantă a acestei legi de învăţare este legea hebbiană a semnalului. În concordanţă cu această lege modificarea ponderilor este dată de:

$$w_{ij}(n+1) = w_{ij}(n) + c\ S(y_i) S(y_j) ,$$

unde S este o **funcţie sigmodală.**

Un alt tip de învăţare este **învăţarea competitivă**. În acest caz, mai mulţi neuroni concurează la privilegiul de a-şi modifica ponderile conexiunilor, adică de a fi activaţi.

12.3. Elemente de neurodinamică

Modelul general al unei reţele neuronale

În concordanţă cu capitolul precedent vom admite că o reţea neurală (RN) constă dintr-o mulţime de elemente de prelucrare (neuroni, unităţi cognitive au noduri ale reţelei) înalt interconectate.

Considerăm in continuare o reţea cu p neuroni.

Aceştia sunt conectaţi printr-o mulţime de ponderi de conexiune sau ponderi sinaptice. Fiecare neuron i are n_i intrări şi o ieşire y_i. Intrările reprezintă semnale venite de la alţi neuroni sau din lumea exterioară.

Intrările x_i ale neuronului i se reprezintă prin numerele reale $x_1^i, \ldots x_n^i$. Fiecare neuron i are n_i ponderi sinaptice, una pentru fiecare intrare a sa. Aceste ponderi se notează cu $w_1^i, w_2^i, \ldots, w_n^i$ şi reprezintă numere reale care pondereazã semnalul de intrare corespunzãtor.

Dacă $w_{ij} > 0$ avem o pondere sinaptică excitatoare, iar dacă $w_{ij} < 0$ avem de-a face cu o pondere inhibitoare. Pentru simplitate, în cele ce urmează se va suprima indicele i. Fiecare neuron calculează starea sa internă sau activarea (excitaţia) totală ca fiind suma ponderată a semnalelor de intrare.

Notând cu S activarea, avem:

$$S = \sum_{j=1}^{n} w_j x_j$$

În ***modelul McCulloch-Pitts*** fiecare neuron este caracterizat de un prag de excitare. Vom nota acest prag cu t atunci ieşirea y a neuronului este +1, dacă activarea totală este egală sau mai mare decât t.

Dacă f : R - R este funcţia de răspuns definită prin:

$$F(x) = \begin{cases} 1, & \text{dacă } x \geq 0 \\ 0, & \text{daca } x < 0, \end{cases}$$

atunci ieşirea neuronului se scrie ca $y = f(\sum w_j x_j + t)$.

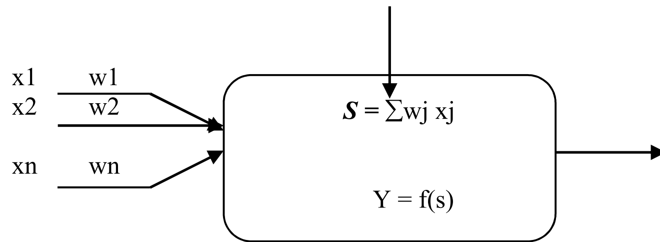
Valoarea prag t poate fi eliminată din argumentul funcţiei f, dacă se adăugă neuronului un semnal de intrare care are întotdeauna valoarea 1 şi ponderea t, adică: $x_{n+1} = 1$, iar $w_{n+1} = t$

În acest caz activarea totală este :

$$S' = \sum_{j=1}^{n} w_j x_j$$

şi ieşirea se poate scrie $y = f(s')$.

Avantajul acestei abordări este acela că pragul poate fi ajustat împreună cu celelalte ponderi în timpul procesului de instruire. Modelul neuronal considerat se poate reprezenta schematic ca in figura următoare.



Forma funcţiei de răspuns f depinde de modelul de reţea neuronala studiat. Aceasta funcţie se mai numeşte funcţie neuronală, funcţie de ieşire sau funcţie de activare a neuronului.

Funcţiile sigmoidale sunt cele mai des folosite deoarece reprezintă forme netezite ale funcţiei prag liniare. Toate aceste funcţii sunt continue, derivabile şi monoton crescătoare. Aceste proprietăţi sunt foarte convenabile pentru aplicaţii.

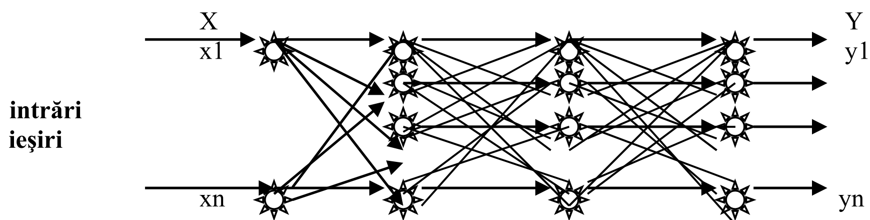
Ştim că funcţiile sigmoidale, ca şi cele cu prag liniar, pot produce şi valori de ieşire intermediare celor două valori extreme (0 şi 1). Acest lucru reprezintă o facilitate pentru calculul analogic şi pentru modelarea unei logici multivalente.

12.4. Reţele neuronale multistrat

Neuronii pot fi conectaţi in diferite moduri pentru a forma o reţea neurală.
Un model uzual de topologie consideră neuronii organizaţi în mai multe straturi.

O reţea neurală multistrat conţine două sau mai multe straturi de neuroni. Primul strat primeşte intrările din mediu. Ieşirile neuronilor din acest strat constituie intrări pentru neuronii stratului următor. Ieşirea reţelei este formată din ieşirile neuronilor ultimului strat. Straturile situate între primul şi ultimul nivel sunt straturi ascunse ale reţelei.

Motivul acestei complicări a arhitecturii este legat de faptul că, uneori arhitecturile mai simple se dovedesc incapabile de a rezolva o problemă sau o clasă de probleme. Dacă o reţea dată nu poate rezolva o problemă, este uneori suficient să mărim numărul neuronilor din reţea, păstrând vechea arhitectură. În alte situaţii, pentru rezolvarea problemei este necesar sa modificam arhitectura reţelei, introducând unul sau mai multe straturi neuronale noi.



În reţeaua din figura de mai sus nu există conexiuni intre neuronii aceluiaşi strat. Semnalul se propagă în reţea dinspre stratul de intrare spre cel de ieşire. Spunem că avem o propagare înainte a semnalului în reţea (reţea cu transmitere înainte a semnalului).
Putem astfel considera arhitecturi de reţea în care există conexiuni între neuronii aceluiaşi strat. De asemenea, uneori poate fi util să considerăm conexiuni de la un neuron spre neuroni aflaţi în stratul anterior (mai apropiat de intrarea reţelei). Alteori, conexiunile pot lega doi neuroni care nu se află neapărat in straturi adiacente.

12.5. Modelul Perceptronului

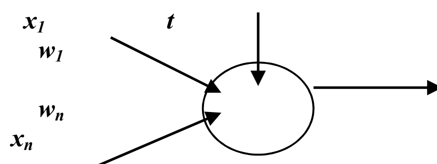
Modelul perceptronului reprezintă sâmburele din care s-au dezvoltat toate celelalte reţele neuronale.

Arhitectura perceptronului standard este cea mai simpla configuraţie posibilă a unei reţele şi ea permite ca instruirea acesteia să se realizeze folosind un algoritm simplu şi eficient.

Algoritmul este reprezentat de o clasă largă de algoritmi de instruire, vectorii din mulţimea de instruire se reprezintă reţelei împreună cu clasa căreia îi aparţin. Dacă au fost memorate suficient de multe obiecte din fiecare clasă, se poate construi o reprezentare internă a fiecărei clase prin ponderile de conexiune ale reţelei.

Perceptronul cu un singur strat

Perceptronul cu un singur strat este cea mai simplă reţea neuronală.
Valorile de intrare pot fi binare sau numere reale. Această reţea elementară are capacitatea de a învăţa să recunoască forme simple. Un perceptron este capabil să decidă, dacă un vector de intrare aparţine uneia din cele două clase de instruire, notate cu A_1 şi A_2.



Acest perceptron este implementat cu un singur neuron. Dacă x este vectorul ce reprezintă intrările neuronului, putem calcula suma ponderată a elementelor de intrare $X = (x_1, x_2, \dots, x_n)^T$, $x_j \in \Re$

$$s = \sum w_I x_I$$

Admiţând că perceptronul are ieşirile –1 şi +1. Ieşirea y este dată de următoarea regulă :

$$s > -t \Rightarrow y = 1,$$
$$s < -t \Rightarrow y = -1,$$

unde t este un prag al neuronului. Ieşirea y = 1 corespunde faptului că vectorul de intrare aparţine clasei A_1.
Dacă ieşirea este –1 atunci vectorul prezentat reţelei aparţine clasei A_2 .
Notând cu x′ vectorul de intrare extins, $x' = (x_1, x_2, \dots, x_n)^T$ şi cu v vectorul ponderilor la care adăugăm o componentă cu pragul t, $v = (w_1, w_2, \dots, w_n, t)^T$.
Cu aceste notaţii ieşirea perceptronului este dată de regula :

$$v^T x' > 0 \Rightarrow y = +1$$
$$v^T x' < 0 \Rightarrow y = -1$$

Ponderile conexinilor şi pragul (componentele vectorului v) unui perceptron se pot adapta folosind un algoritm de instruire simplu numit **algoritmul perceptronului.**

Algoritmul de instruire

Se va prezenta algoritmul standard pentru instruirea perceptronului cu două clase de instruire.
La primul pas al algoritmului de instruire se iniţializează valorile ponderilor şi pragul. Ponderile iniţiale sunt componentele vectorului v^1. Se aleg pentru ponderile iniţiale numere reale mici şi nenule şi alegem valoarea constantei de corecţie c (de obicei $0 < c \leq 1$).

Constanta c controlează rata de adaptare a reţelei. Alegerea ei trebuie să satisfacă cerinţe contradictorii: pe de o parte există necesitatea de a obţine o adaptare rapidă a ponderilor la intrări şi pe de altă parte trebuie să se ţină seama de necesitatea ca medierea intrărilor precedente să genereze o estimare stabilã a ponderilor.

Algoritmul de învăţare al perceptronului compară ieşirea produsă de reţea cu clasa corespunzătoare a vectorului de intrare. Dacă s-a produs o clasificare eronată, atunci vectorul pondere este modificat. În caz contrar el rămâne neschimbat.

Dacă la p paşi consecutivi nu se produce nici o modificare a vectorului pondere, atunci toţi vectorii (formele) de instruire sunt corect clasificaţi de ultimul vector ponderere rezultat. Am obţinut un vector soluţie şi algoritmul se opreşte cu această decizie. Procedura de instruire descrisă mai sus conduce la algoritmul următor pentru instruirea perceptronului .

Algoritmul perceptronului

P1 **Se iniţializează ponderile şi pragul.**
Se aleg componentele $w_1, \ldots\ldots, w_n$ şi t ale vectorului v^1.
Se pune k = 1 (v^1 este vectorul pondere la momentul iniţial).

P2 **Se alege constanta de corecţie c > 0.**

P3 **Se alege o nouă formă de instruire.**
Se reprezintă reţelei o nouă formă de intrare z^k şi ieşirea dorită, (aşteptată) corespunzătoare.

P4 **Se calculează ieşirea reală generată de perceptron. Ieşirea reală este dată de semnul expresiei $(v^k)^T z^k$.**

P5 **Condiţia de oprire**
Se repetă pasul P6 până când vectorul pondere nu se modifică la p paşi consecutivi.

P6 **Se adaptează ponderile şi pragul**.
Se modifică vectorul pondere folosind regula de corecţie:

$$\begin{cases} v^k + cz^k, & \text{dacă } (v^k)^T z^k \leq 0, \\ v^k, & \text{dacă } (v^k)^T z^k > 0 . \end{cases}$$

Se pune k = k + 1.
Dacă algoritmul s-a oprit normal, atunci vectorul v^{k+1} reprezintă o soluţie a problemei de instruire .

Limitele perceptronului

În multe probleme concrete de clasificare şi de instruire intervin clase de obiecte care nu sunt liniar separabile. Deoarece perceptronul nu poate discrimina decât clase liniar separabile, aplicarea acestui algoritm în rezolvarea unor probleme concrete este sever limitată.

Cea mai celebră şi una dintre cele mai simple probleme care nu pot fi rezolvate de un perceptron este, problema calculării valorilor funcţiei logice **sau exclusiv.**

Problema poate fi rezolvată de un perceptron cu mai multe straturi (cel puţin cu două straturi).

Această limitare, nu se datorează algoritmului, ci este legată de topologia foarte simplă reţelei utilizate. Dacă problema de instruire necesită regiuni de decizie mai complicate, atunci trebuie mărită complexitatea reţelei.

Propagarea înapoi a erorii

Algoritmul de propagare înapoi (Back Propagation) este considerat in mod uzual, ca fiind cel mai important şi mai utilizat algoritm pentru instruirea reţelelor neuronale.

Algoritmul de propagare înapoi este o metodă de instruire în reţelele neuronale multistrat cu transmitere înainte (reţele unidirecţionale), în care se urmăreşte minimizarea erorii medii pătratice printr-o metodă de gradient.

Caracteristica esenţială a reţelelor cu două straturi este că ele proiectează forme de intrare similare în forme de ieşire similare fapt ce permite să facă generalizările rezonabile şi să prelucreze acceptabil forme care nu li s-au mai prezentat niciodată.

12.6. Funcţia Criteriu

Tehnica de instruire descrisă în continuare este asemănătoare cu cea utilizată pentru a găsi dreapta care aproximează cel mai bine o mulţime dată de puncte (problema regresiei). Această tehnică este o generalizare a metodei **MEP**. Pentru determinarea dreptei de regresiei, se utilizează, de regulă, **metoda celor mai mici pătrate**. Deoarece este plauzibil ca funcţia pe care o căutăm să fie neliniară, se foloseşte o variantă iterativă a metodei celor mai mici pătrate.

Fie $f : \Re^n \rightarrow \Re^p$ funcţia necunoscută pe care reţeaua trebuie să înveţe să o realizeze.

Vectorul de intrare x^i îi corespunde răspunsul dorit $d^i = f(x^i)$.

La momentul k, reţelei i se reprezintă asocierea vectorială (x^k, d^k). Admiţând că mulţimea de instruire e formată din m asocieri: $(x^1,d^1), \dots ,(x^m,d^m)$ starea curentă (la momentul k) a reţelei neuronale defineşte funcţia vectorială: $N^k : R^n \rightarrow R^p$, $y = N^k(x)$, unde y este vectorul care indică activarea neuronilor ultimului strat al reţelei $y = (y_1,y_2, \dots y_p)^T$.Dacă S este funcţia vectorială de ieşire, atunci la momentul k, reţeaua generează răspunsul $S(y^k)$. Ieşirea vectorială a reţelei este, aşadar : $N^k(x^k) = S(y^k)$.

Notâm cu S_j:R→R funcţia de răspuns a neuronului j din stratul final. Funcţiile S_1,.., S_p sunt mărginite şi crescătoare.

Se poate scrie deci sub forma:

$$N^k(x^k) = \begin{bmatrix} S_1(y_1^k) \\ \dots \\ S_p(y_p^k) \end{bmatrix}$$

Algoritmul de propagare înapoi cere ca funcţiile de răspuns să fie derivabile. Această condiţie elimină posibilitatea utilizării funcţiilor cu prag liniar, nederivabile în punctele prag. Rezultă, deci, că se poate folosii funcţia sigmoidală sau funcţia identică.

O funcţie de răspuns liniară defineşte un neuron de ieşire liniar.

Ieşirea reală a neuronului de ieşire j, când la intrare se prezintă vectorul x^k este: $o_j^k = S_j^o(y_j^k)$, unde indicele o desemnează ieşirea.

Eroarea corespunzătoare neuronului j când reţelei i se furnizează vectorul x^k este:

$$e_j^k = d_j^k - o_j^k = d_j^k - S_j^o(y_j^k).$$

Eroarea pătratică a tuturor neuronilor de ieşire in raport cu intrarea x^k se defineşte ca

$$E_k = 1/2 \sum (e_j^k)^2$$

Eroarea pătratică se mai poate scrie :

$$E_k = 1/2 (e^k)^T e^k.$$

Eroarea totală este suma erorilor pentru cele m asocieri din mulţimea de învăţare:

$$E = \sum_{k=1}^{m} E_k$$

unde m este numărul de forme din această mulţime.

Algoritmul de propagare înapoi urmăreşte minimizarea erorii pătratice, însă nu este sigur că algoritmul găsit va minimiza şi această funcţie criteriu. Ceea ce se obţine este, în general, o soluţie acceptabilă şi nu, neapărat, o soluţie optimă.

12.7. Algoritmul de propagare înapoi

Aplicarea algoritmului de propagare înapoi presupune două etape. În prima etapă se prezintă reţelei un vector de intrare x^k şi ieşirea dorită corespunzătoare. Fie d^k această ieşire, semnalul de intrare se propagă prin reţea şi generează un răspuns.

Neuronul j din câmpul de ieşire produce semnalul: $o_j^k = S_j^o(y_j^k)$.

Acest semnal este comparat cu răspunsul dorit d_j^k.

Rezultă un semnal de eroare δ_{kj} care se calculează pentru fiecare neuron de ieşire:

$$\delta_{ij} = (d_j^k - o_j^k)\frac{\partial S_j^0}{\partial y_j^k}, \quad j=1,2,\ldots p.$$

Se calculează schimbările ponderilor pentru toate conexiunile ce intră în stratul final, folosind regula următoare :

$$V_{qj}^0(k+1) = V_{qj}^0(k) + c\delta_{kj} S_q^h(h_q^k),$$

unde $S_q^h(h_q^h)$ este ieşirea neuronului ascuns q.

Cea de a doua fază a algoritmului corespunde propagării înapoi prin reţea a semnalului de eroare. Aceasta permite calculul recursiv al erorii conform formulei:

$$\delta_{qk}^h = \frac{\partial S_q^k}{\partial h_q^k}(h_q^k)\,\delta_{kj} v_{qj}^0$$

Calculul se face pentru fiecare neuron q din stratul ascuns. Altfel, eroarea se propagă înapoi prin reţea. Propagarea înapoi implică un calcul de aceeaşi complexitate cu propagarea înainte a semnalului şi nu necesită un efort excesiv de timp şi memorie.

12.8. Arhitecturi moderne de reţele neuronale

REŢELE NEURONALE PROBABILISTICE

Modelarea reţelelor Neuronale cu ajutorul teoriei probabilităţilor sau a **teoriilor de incertitudine** aduce o serie de avantaje faţă de abordările strict deterministe. Acestea sunt:

- reprezentarea mai veridică a modului de funcţionare a reţelelor neuronale biologice, în care semnalele se transmit mai ales ca impulsuri;
- eficienţă de calcul mult superioară celei din cadrul reţelelor neuronale, înainte-înapoi';
- implementare hardware simplă pentru structuri paralele;
- instruire uşoară şi cvasiinstantanee, rezultând posibilitatea folosirii acestor reţele neuronale în timp real;
- forma suprafeţelor de decizie poate fi oricât de complexă prin modificarea unui singur parametru (de netezire) s, aceste suprafeţe putând aproxima optim suprafeţele Bayes;
- pentru statistici variabile în timp, noile forme pot fi suprapuse simplu peste cele vechi;
- comportare bună la forme de intrare cu zgomot sau incomplete.

Astfel, pentru o problemă biclasă, considerăm o reţea neuronală probabilistică în care stratul numit ,**unităţi ale formei** conţine elemente având structura din figura de mai jos.

Pentru o **distribuţie normală a densităţii de probabilitate**, estimatorul pentru clasa A este:

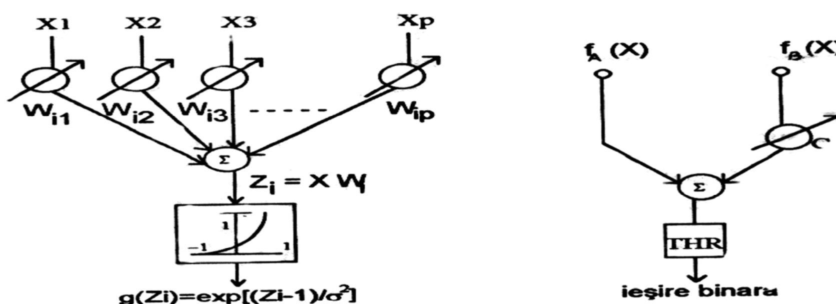
$$f_A\ (x)\ = \frac{1}{(2\pi)^{p/2}}\frac{1}{\sigma^p}\frac{1}{m}\sum_{i=1}^{n} \exp\left|-\frac{(X-X_{Ai})^t(X-X_{Ai})}{2\sigma^2}\right|$$

funcţie ce apare la ieşirea unităţii formei, unde f_A : X→W. Unităţile sumatoare adună datele de la unităţile formei ce corespund mulţimii de instruire şi au o pondere variabilă, C_k :

$$C_k = -(h_{Bk}\, I_{Bk})/(h_{Ak}\, I_{Ak}) *(\, n_{Ak} / n_{Bk}),$$

unde n_{Ak} şi n_{Bk} sunt numărul formelor de antrenare pentru clasa A_k respectiv B_k , h sunt probabilităţile a priori, I sunt funcţii de pierdere în cazul unei decizii eronate, iar THR este funcţia prag.

Instruirea constă în identitatea dintre vectorul Wi şi fiecare formă de antrenare x. Astfel, este necesară câte o unitate pentru fiecare formă din mulţimea de instruire.



REŢELE NEURONALE FUZZY

Combinarea celor două clase de sisteme cognitive : nuanţate (fuzzy) şi neuronale, s-a impus prin performanţele bune, de câţiva ani. Fuzzyficarea arhitecturilor neuronale măreşte capacitatea reţelelor neuronale de a recunoaşte (clasifica) forme complexe, distorsionate sau incomplete.

O structură de reţea neuronală a fost propusă de către **Watanabe** . Acest model foloseşte **neuronul logic.**

Neuronul logic are avantajul vitezei şi al unei capacităţi discriminatorii remarcabile (pragul fiind independent de mărimea în biţi a componentelor vectorului de intrare). Acum avem două tipuri de vectori pondere:

$$W_p = (W_{p1}, \ldots, W_{pn}) \text{ şi } W_n = (W_{n1}, \ldots, W_{nn})$$

Răspunsul analog y, ce urmează a fi cuantizat, are forma:

$$y = \wedge (x_i\ \alpha_i)$$

unde x_I fiind intrările. Ieşirea binară a sistemului este :

$$z = SGN(y\text{-}h),$$

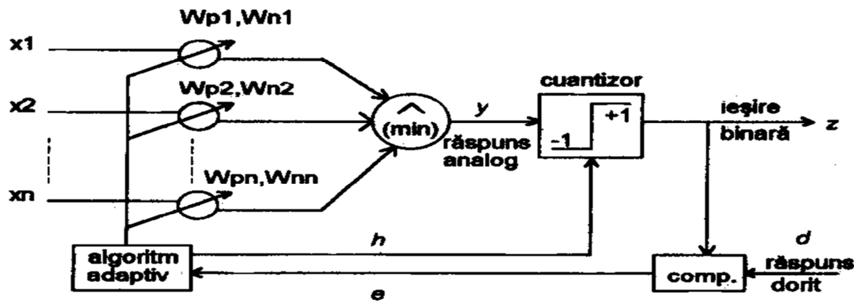
unde h este **pragul fixat.**

Vectorul pondere este ajustat după diferenţa e între ieşirea dorită şi răspunsul sistemului la diferite intrări în cursul instruirii iar h se calculează cu operaţii logice fuzzy:

$$h_t = (\vee W_{pi}) \vee (w_{ni})$$

$$h_B = (W_{pi}) \wedge (W_{ni})$$

$$h = (h_T + h_B)/n$$



12.9. Prelucrări de imagini cu reţele neuronale

Proprietăţile de bază ale reţelelor neuronale, anume faptul ca sunt aproximatori universali şi ca au o capacitate de predicţie deosebită, îşi găsesc utilizarea imediată in cadrul altor domenii, cum ar fi: prelucrarea de imagini, recunoaşterea formelor vizuale(scris şi amprente) sau recunoaşterea vorbirii.

Filtrarea zgomotelor, accentuarea şi detecţia contururilor, segmentarea imaginii şi extragerea caracteristicilor sunt cele mai importante operaţii de procesare. Deşi metodele clasice de prelucrare satisfac majoritatea aplicaţiilor există tehnici hibride de tipul segmen-

tare/clasificare, pentru care metodele conexioniste oferă performante superioare in raport cu rata de recunoaştere şi cu timpul de calcul.

Implementarea se poate realiza cu reţele neuronale celulare, care sunt circuite analogice neliniare ce permit o implementare VLSI, reţelele celulare putând efectua prelucrări paralele de semnal in timp real. Ecuaţia de ieşire a unui nod oarecare reprezintă un filtru bidimensional, neliniar şi invariant în spaţiu.

O aplicaţie foarte utilă este recunoaşterea **on line** a semnăturilor grafologice cu ajutorul unui neurocalculator. Se pot folosi drept caracteristici coordonatele planare şi presiunea stiloului, extrase cu ajutorul unei tabele grafice. Sunt luate în consideraţie variabilitatea semnaturilor corecte, precum si semnaturile false.

Vom prezenta în continuare o aplicaţie care se bazează pe reţele Neuronale de tip **Hopfield** pentru recunoasterea formelor intr-o imagine binara.

Pavlidis propune clasificarea imaginilor în patru clase, astfel:

- **Clasa (1)** de imagini include imaginile în scară de gri (sau color).
- **Clasa (2)** de imagini cuprinde imaginile binare (sau cu câteva culori).
- **Clasa (3)** de imagini include imagini formate din linii si curbe continue.
- **Clasa (4)** de imagini cuprinde imagini formate din puncte izolate sau poligoane.

Se remarcă scăderea complexităţii imaginii odată cu numărul clasei, simultan cu reducerea semnificativă a volumului de date necesar pentru stocarea lor (**Howe**).

Reţeaua neurală Hopfield

Acest tip de reţea poate fi folosit ca memorie asociativă sau pentru rezolvarea unor probleme de optimizare. Foloseşte neuroni cu intrări binare, a căror ieşire conţine nelinearităţi de tipul limitare hardware. Ieşirea fiecărui neuron este aplicată tuturor celorlalte noduri prin intermediul unor ponderi.

J. J. Hopfield în lucrarea ***Neural networks and physical systems with emergent collective computational abilities. Proc.Natl. Acad. Sci. 79 1982,*** a demonstrat că această reţea converge dacă ponderile sunt simetrice. Desi simplitatea ei o face atractivă, această reţea prezintă două limitări majore când este
utilizată ca memorie adresabilă prin conţinut. În primul rând, numărul de forme prototip care pot fi stocate si apoi regăsite corect în reţea este limitat la aproximativ 15% din numărul de neuroni.

Dacă se încearcă stocarea prea multor forme prototip, reţeaua ar putea converge către un prototip fictiv. A doua limitare prezentă la reţeaua Hopfield apare, dacă prototipurile memorate în reţea sunt foarte asemănătoare (prezintă un mare număr de biţi identici), caz în care reţeaua devine instabilă.

Dacă reţeaua Hopfield este folosită drept clasificator, ieşirea reţelei (după convergenţă) trebuie comparată cu fiecare formă prototip, după care se poate lua decizia de apartenenţă la una din clasele ataşate formelor prototip.

Algoritmul de antrenare al retelei Hopfield:

Pasul 1. Iniţializarea ponderilor conexiunilor:

$$t_{ij} = \sum_{s=0}^{M-1} x_j^s x_i^s, \text{ dacă } i \neq j; \; 0 \text{ dacă } i=j, \; 0 \text{ £ } i \text{ £ } M-1, \; 0 \text{ £ } j \text{ £ } M-1,$$

unde t_{ij} este ponderea conexiunii de la nodul „i” la nodul „j”.

Pasul 2. Se aplică un vector de intrare necunoscut:

$$\mu_j\,(t+1)=f_h\sum_{i=0}^{N-1} t_{ij}\,u_j(t),\ 0 \pounds j \pounds\ M\text{-}1$$

unde funcţia neliniara f_h este de tipul limitare hardware. Operaţia se repetă până când toate ieşirile neuronilor rămân neschimbate. În acest caz, configuraţia ieşirilor coincide cu forma prototip cea mai apropiată de forma necunoscută de intrare.

Pasul 3. Iterează până la convergenţă.

Pasul 4. Repeta paşii 2-3 pentru orice altă formă necunoscută de intrare.

Aplicaţia. Să se construiască o reţea neurală Hopfield pentru a recunoaşte formele dintr-o imagine binară.
Cu ajutorul butonului "Încarcă Imagine" se încarcă o imagine binară , învaţarea acestei imagini se face cu ajutorul butonului "încarcare RN". Formele învăţate vor apărea în partea de jos a aplicaţiei , în secţiunea "imagini încarcate".
Pentru a testa reteaua se poate încarca orice imagine binară sau se poate adăuga zgomot. Se poate alege orice nivel de zgomot de la 1% la 100%.
Apoi se apasă butonul "Execută" pentru a testa forma nouă.

Codul Matlab

```
function varargout = hopfieldNetwork(varargin)
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @hopfieldNetwork_OpeningFcn, ...
                   'gui_OutputFcn',  @hopfieldNetwork_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
function hopfieldNetwork_OpeningFcn(hObject, eventdata, handles,
varargin)

handles.output = hObject;
N = str2num(get(handles.imageSize,'string'));
handles.W = [];
handles.hPatternsDisplay = [];
guidata(hObject, handles);
function varargout = hopfieldNetwork_OutputFcn(hObject, eventdata,
handles)
varargout{1} = handles.output;
function reset_Callback(hObject, eventdata, handles)
    for n=1 : length(handles.hPatternsDisplay)
        delete(handles.hPatternsDisplay(n));
    end
```

```
    handles.hPatternsDisplay = [];
     set(handles.imageSize,'enable','on');
    handles.W = [];
    guidata(hObject, handles);
function imageSize_Callback(hObject, eventdata, handles)
    num = get(hObject,'string');
    n = str2num(num);
    if isempty(n)
        num = '32';
        set(hObject,'string',num);
    end
    if n > 32
         warndlg('este recomandat sa nu se lucreze cu retele mai mari
decat 32^2 neuroni!','!! Atentie !!')
    end
function imageSize_CreateFcn(hObject, eventdata, handles)
if ispc
    set(hObject,'BackgroundColor','white');
else
set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'
));
end
% executa încarcarea imaginii
function loadIm_Callback(hObject, eventdata, handles)
     [fName dirName] = uigetfile('*.bmp;*.tif;*.jpg;*.tiff');
    if fName
        set(handles.imageSize,'enable','off');
        cd(dirName);
        im = imread(fName);
        N = str2num(get(handles.imageSize,'string'));
        im = fixImage(im,N);
        imagesc(im,'Parent',handles.neurons);
        colormap('gray');
    end
% --- se execută la apasarea invatarii
function train_Callback(hObject, eventdata, handles)

    Npattern = length(handles.hPatternsDisplay);
    if Npattern > 9
        msgbox('mai mult de 10 parti paternale nu suport!','eroare');
        return
    end
    im = getimage(handles.neurons);
    N = get(handles.imageSize,'string');
    N = str2num(N);
    W = handles.W;
    avg = mean(im(:));
    if ~isempty(W)
         W = W + ( kron(im-avg,im-avg))/(N^2)/avg/(1-avg);
    else
        W = ( kron(im-avg,im-avg))/(N^2)/avg/(1-avg);
    end
      ind = 1:N^2;
    f = find(mod(ind,N+1)==1);
    W(ind(f),ind(f)) = 0;
      handles.W = W;
        % înlocuieste noua imagine
```

```
    xStart = 0.01;
     xEnd = 0.99;
    height = 0.65;
    width = 0.09;
    xLength = xEnd-xStart;
    xStep = xLength/10;
    offset = 4-ceil(Npattern/2);
    offset = max(offset,0);
    y = 0.1;
    if Npattern > 0
        for n=1 : Npattern
            x = xStart+(n+offset-1)*xStep;
            h = handles.hPatternsDisplay(n);
            set(h,'units','normalized');
            set(h,'position',[x y width height]);
        end
         x = xStart+(n+offset)*xStep;
        h = axes('units','normalized','position',[x y width height]);
        handles.hPatternsDisplay(n+1) = h;
        imagesc(im,'Parent',h);
    else
        x = xStart+(offset)*xStep;
        h = axes('units','normalized','position',[x y width height]);
        handles.hPatternsDisplay = h;
    end
     imagesc(im,'Parent',h);

set(h,'YTick',[],'XTick',[],'XTickMode','manual','Parent',handles.lea
rnedPaterns);
    guidata(hObject, handles);
 % --- se execută la apăsarea adaugă zgomot.
function addNoise_Callback(hObject, eventdata, handles)
    im = getimage(handles.neurons);
    noisePercent = get( handles.noiseAmount, 'value' );
    N = round( length(im(:))* noisePercent );
    N = max(N,1);
    ind = ceil(rand(N,1)*length(im(:)));
    im(ind) = ~im(ind);
    imagesc(im,'Parent',handles.neurons);
     colormap('gray');
% --- se executa la apasarea butonului execută.
function Executa_Callback(hObject, eventdata, handles)
    im = getimage(handles.neurons);
    [rows cols] = size(im);
    if rows ~= cols
        msgbox('nu suport nici un patratel','eroare');
        return;
    end
     N = rows;
    W = handles.W;
    if isempty(W)
        msgbox('nici o imagine incarcata','eroare');
        return;
    end
    mat = repmat(im,N,N);
    mat = mat.*W;
    mat = im2col(mat,[N,N],'distinct');
```

```
    networkResult = sum(mat);
     networkResult = reshape(networkResult,N,N);
     im = fixImage(networkResult,N);
     imagesc(im,'Parent',handles.neurons);
function im = fixImage(im,N)
    if length( size(im) ) == 3
        im = rgb2gray(im);
    end
    im = double(im);
    m = min(im(:));
    M = max(im(:));
    im = (im-m)/(M-m);
    im = imresize(im,[N N],'bilinear');
    im = (im > 0.5);

% --- se executa la mişcarea sliderului
function noiseAmount_Callback(hObject, eventdata, handles)
    percent = get(hObject,'value');
    percent = round(percent*100);
    set(handles.noisePercent,'string',num2str(percent));
function noiseAmount_CreateFcn(hObject, eventdata, handles)
usewhitebg = 1;
if usewhitebg
    set(hObject,'BackgroundColor',[.9 .9 .9]);
else
set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'
));
end
```

Reţele neuronale de tip Kohonen

T.Kohonen ***în lucrarea Self-organized formation of topologically correct features maps, Biol. Cybern. 43, pp. 59-69 , 1982*** propune un nou tip de reţele Neuronale cu două straturi: unul de intrare şi altul de ieşire . Toate intrările sunt conectate la toate nodurile de ieşire , care pot avea numai conexiuni laterale cu nodurile imediat vecine. Reţelele Kohonen sunt cu autoorganizare, de tip feedforward şi cu instruire nesupervizată şi se folosesc pentru gruparea datelor de intrare de tip ***Clustering.***

Reţelele Kohonen sunt destinate în principal clasificării nesupervizate a datelor vectoriale, asigurând în plus faţă de celelalte reţele competitive şi conservarea relaţiilor de vecinătate din domeniul datelor de intrare. Aceasta înseamnă că datele de intrare similare vor fi în acelaşi clasă fie în clase diferite, dar care sunt reprezentate de către unităţi funcţionale vecine.

MatLab prin ***toolboxul de Neural Network*** permite crearea de hărţi multidimensionale cu diferite tipuri de topologii ale nivelului de ieşire. Conectivitatea dintre nivelul de intrare şi cel de ieşire este totală, unitatea învingătoare se determină folosind criteriul dinstanţei minime. Antrenarea este de tip conectiv asigurându-se ajustarea atât a unităţii învingătoare cât şi a celor aflate în vecinătatea acesteia (folosind însă o rată de învăţare de două ori mai mică). Atât rata de învăţare cât şi dimensiunea vecinătăţii descresc pe măsură ce numărul de iteraţii creşte.

Învăţarea conţine două etape:

- *Etapa de ordonare-* asigură ordonarea unităţilor în aşa fel încât unităţi vecine să răspundă de clase apropiate (în aceasta fază atât rata de învăţare cât şi dimensiunea vecinătăţii sunt relative mari).

- *Etapa de ajustare* fină a prototipurilor: asigură apropierea vectorilor cu ponderi de prototipurile claselor (vecinăţile se reduc doar la vecinii direcţi sau chiar doar la unitatea învingătoare, iar rata de învăţare este mică).

Pentru crearea unei reţele Kohonen ***Matlab*** pune la dispoziţia utilizatorilor funcţia: **Newsom** ([min1 max1; ...; minN maxN],[dim1,dim2,...], grila, dist, rata1, epoci1, rata2, vecinmin) cu următoarele semnificaţii pentru parametri:

- **min i şi max i**: limitele domeniului datelor de intrare în unitatea i.
- **dim1**, **dim2**, … dimensiunile nivelului cu unităţi funcţionale. Pentru o reţea unidimensională se specific doar dim1, pentru una bidimensională se specific dim1, dim2 etc.
- **grila**: specifică structura grilei. Valori posibile: gridtop (pătratică), hextop (hexagonală),

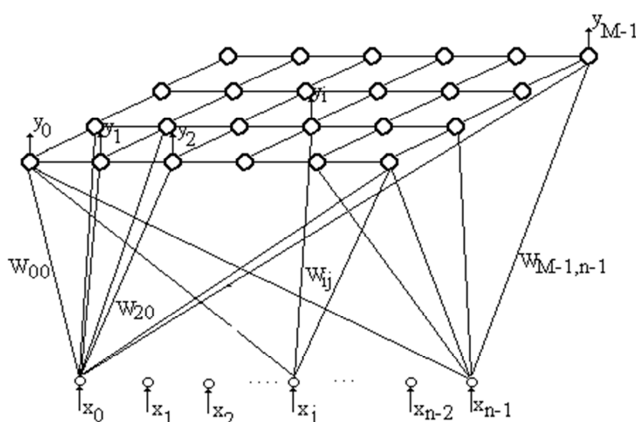
randtop (grilă cu structură aleatoare). Valoarea implicită este hextop.

- **dist**: tipul distanţei utilizate în stabilirea vecinătăţii. Valori posibile: dist (distanţa euclidiană), mandist (distanţa Manhattan), boxdist şi linkdist (distanţe bazate pe numărul de "paşi" ce trebuie efectuaţi în cadrul grilei pentru a ajunge de la un nod la altul). Valoare implicita: linkdist.
- **rata1**: rata de învăţare în faza de ordonare. Valoare implicită: 0:9.
- **epoci1**: număr maxim de epoci în faza de ordonare. Valoare implicită: 1000.
- **rata2**: rata de învăţare în faza de ajustare. Valoare implicită: 0:02.
- **vecinmin**: dimensiunea vecinătăţii în faza de ajustare. Valoare implicită: 1.

Din punct de vedere al timpului când se face actualizarea ponderilor reţelei există două tipuri de învăţare :

- *off-line* se determină pentru fiecare intrare-ieşire, modificarea care trebuie adusă coeficienţilor sinaptici. Aceste modificări se sumează şi se aplică reţelei numai după ce au fost prezentate toate datele de antrenament (perechile de intrare-ieşire).
- *On-line* modificarea coeficienţilor, calculaţi pentru o pereche de intrare –ieşire se aplică reţelei imediat după prezentarea acestei perechi. Prezintă în raport cu precedentul, avantajul că este în general mai rapid şi poate părăsii unele minime locale ale funcţiei de eroare totală.

Prezentăm în continuare o reţea neurală **Kohonen** rectangulară:



Unde:
- $(X_0,X_1,\ldots X_{N-1}) \rightarrow$ lotul de vectori de clasificat
-$X = (x_0,x_1,\ldots x_{n-1})^t \rightarrow$vector de intrare aparţinând lotului de clasificat
-$w_{ij} \rightarrow$ponderea reţelei care face legătura între neuronal de intrare j şi neuronal ieşire i
-i=0,…,M-1,unde M numărul de neuroni din stratul de ieşire
-j=0,…,n-1, unde n-dimensiunea vectorului de intrare.

Algoritmul de rafinare a ponderilor reţelei

Se consideră lotul de forme (vectori) de clasificat de forma $(X_0,X_1,\ldots X_{N-1})$, unde N-numărul de vectori din lot. Algoritmul de rafinare se bazează pe principiul vecinătăţii, rafinarea făcându-se atât pentru neuronal câştigător cât şi pentru vecinii săi. Vecinătatea unui neuron se va iniţializa la o valoare mai mare (de obicei egală cu dimensiunea reţelei), după care ea scade pe parcursul rafinării astfel încât să se reducă la un singur neuron. De obicei vecinătatea este constantă pe parcursul unei parcurgeri a lotului de vectori, modificarea ei facându-se după fiecare iteraţie de lot. Legea de variaţie a vecinătăţii se allege experimental funcţie de aplicaţie, în general aceasta scăzând după fiecare iteraţie cu doi neuroni pe fiecare dimensiune.

Paşi:

1. Iniţializare ponderilor reţelei se face aleator într-un interval de valori ce conţine cont de valoarea vectorilor de intrare

$w_{ij}(t=0)$, i=0, …M-1, j=0,…n-1.
Precum şi stabilirea dimensiunii iniţiale a vecinătăţii V(0).

2. t=t+1, unde t=kn+p unde
   t - numărul iteraţiei curente
   k- numărul de iteraţii de lot (epoci) parcurse
   p- indicele vectorului curent X_p, p=0,…,N-1 care se aplică la intrarea reţelei vectorului curent.
3. Se calculează distanţele $d_i = SUM[x_j(t)- w_{ij}(t)]^2$
4. Se calculează neuronul i^* (neuronal câştigător) astfel încât distanţa d_i^* să fie minimă;
5. Se rafinează ponderile aferente neuronului i^* şi neuronilor aparţinând vecinăţii $V_i^*(t)$ astfel:

$w_{ij}(t+1)= w_{ij}(t) +\rho(t)[x_j(t)- w_{ij}(t)]$, unde $\rho(t) \in (0,1)$ este rata de învăţare.

6. -Dacă nu s-a terminat parcurgerea lotului de vectori se trece la pasul doi,
   -Dacă s-a terminat de parcurs tot lotul de vectori se verifică condiţia de stop, altfel se reia pasul 2.

Reguli de stop:

a. În general rafinarea reţelei se opreşte când gradientul de modificare a ponderilor reţelei devine nesimnificativ.
b. Se mai poate oprii algoritmul şi după un număr fixat de iteraţii dat fiind că rafinarea reţelei este un process de durată.

SOM (Self-Organizing-Maps)

Este o reţea de învăţare nesupervizată în care unităţile de intrare-ieşire sunt dispuse sub formă de grilă (în general rectangulară) în plan, iar coeficienţii conexiunilor între stratul de ieşire depend de distanţele între unităţi. Astfel dacă unităţile sunt apropiate acestea se inflenţează reciproc, iar dacă sunt depărtate influenţa este inhibatoare. Se defineşte astfel pentru fiecare neuron o vecinatate a sa. Forma acestei vecinătăţi poate varia în funcţie de datele de intrare şi poate fi rectangulară sau circulară . În timpul învăţării competitive se actualizează nu numai vectorul asociat unităţii învingătoare ci şi (uneori cu un coeficient mai redus) unităţilor din vecinătatea acesteia. Acest mod de modificare încurajează unităţile vecine să răspundă în mod similar la vectori de intrare similari.

Reţeua este utilizată în aproximarea distribuţiei vectorilor de intrare, în reducerea dimensionalităţilor datelor menţinând pe cât este posibil vecinătatea şi în clustering. Arhitectura reţelei SOM constă într-o matrice bidimensională de unităţi de ieşire, fiecare conectată cu toate cele n unităţi de intrare.

Fie w_{ij} vectorul n×n bidimensional asociat . Fiecare neuron de ieşire calculează distanţa dintre vectorul de intrare x şi vectorul pondere w_{ij} memorat. Ponderea vectorului învingător şi a celor din vecinătatea acestuia după **formula lui Kohonen.**

Ideea acestui algoritm este că se caută unitatea învingătoare pentru fiecare vector de intrare, dar modificarea coeficienţilor sinaptici, dar modificarea coeficienţilor sinaptici se realizează nu numai pentru unitatea învingătoare ci şi cu un coeficient mai redus şi pentru unităţile aflate în vecinătatea unităţii învingătoare.

Acest mod de modificare a coeficienţilor încurajează unităţile vecine să răspundă în mod similar la vectori de intrare similari ⇒ reţeaua este astfel o hartă a mulţimii vectorilor de intrare.

Algoritmul de învăţare Kohonen

Ideea de învăţare constă în a plasa reţeaua undeva în plan şi a o antrena la început cu o vecinătate şi a o antrena la început cu o vecinătate şi un coeficient de învăţare mai mare care pe parcursul învăţării să scadă treptat .

Paşi:

1. Se dispun unităţile de ieşire sub formă de grilă în plan (această dispunere reprezintă de fapt şi iniţializarea ponderilor unităţilor cu valori aleatoare mici).

Se recomandă memorarea acestor unităţi într-o matrice pentru a putea calcula ulterior vecinătatea unui neuron. În cazul reţelei SOM ponderea unui neuron este considerată poziţia acelui neuron în spaţiul de reprezentare al datelor de intrare.

2. Se stabileşte o anumită formulă pentru calculul vecinătăţii şi al coeficientului de învăţare , formulă care să depindă de pasul la care s-a ajuns în procesul de învăţare.
3. Se ia un vector de intrare şi se calculează distanţa dintre el şi toţi neuronii din reţea. Neuronul care este cel mai apropiat este considerat neuronul învingător şi la acesta i se modifică ponderea, la toţi ceilalţi li se recalculează ponderea după aceaşi formulă.
4. Dacă coeficientul de învăţare a ajuns la zero se consideră că reţeaua a învăţat, deoarece nu vor mai avea loc modificări în reţea . Dacă coeficientul de învăţare nu a ajuns la zero se sare la pasul 3.

Aplicaţie 1

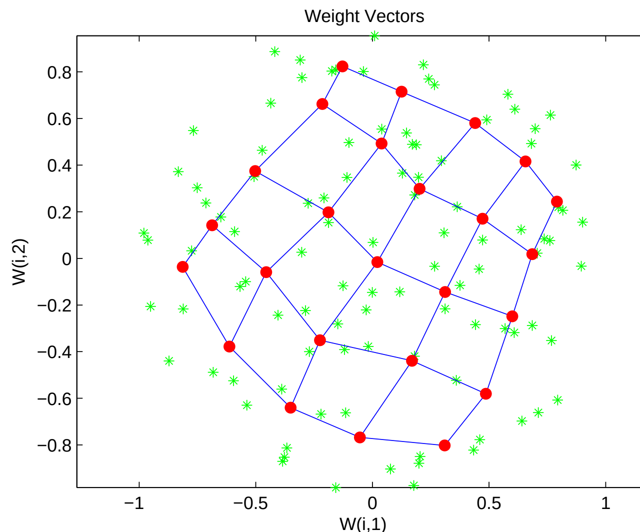
Să se simuleze o hartă Kohonen unidimensională, datele de intrare generate aleator in [0,1]X[0,1]:

Codul Matlab

```
date=rand(2,100);
% definirea retelei (cu doua unitati de intrare si grila 5*5)
som=newsom([0 1;0 1],20);
som.trainParam.epochs=1000;
% reprezentarea datelor
plot (date(1,:),date(2,:),'g*')
hold on; % pastreaza graficul
soma=train(som,date);
% reprezentarea grafica
plotsom(soma.iw{1,1},soma.layers{1}.distances);
hold off;
```

Aplicaţie 2.
Să se simuleze o hartă Kohonen bidimensională, datele de intrare generate aleator in C((0,0),1)

Codul Matalab
```
for i=1:100
x=2*rand-1; y=2*rand-1;
while (x^2+y^2)>1
x=2*rand-1; y=2*rand-1;
end
date(1,i)=x; date(2,i)=y;
end;
% definirea retelei (cu doua unitati de intrare si grila p¸atratic¸a
5*5)
som=newsom([-1 1;-1 1],[5 5],'gridtop');
som.trainParam.epochs=1000;
plot (date(1,:),date(2,:),'g*');
hold on; % pastreaza graficul
soma=train(som,date);
% reprezentarea grafica
plotsom(soma.iw{1,1},soma.layers{1}.distances);
hold off;
```
În urma execuţiei programului obţinem rezultatul:



Observaţie

1. Matalb-ul pune la dispoziţia utilizatorilor un toolbox : ***Neural Network Toolbox*** ultraspecializat cu ajutorul căruia se pot face simulări eficiente : astfel crearea şi vizualizarea unei reţele **Kohonen** se poate face folosind funcţia `nctool` şi un set de date de test.
2. Recomand celor interesaţi de aprofundarea acestor reţele Neuronale următoarele lucrări de referinţă :
   - **B.Valuru** – ***Neural Networks and Fuzzy Logic, 2006,***
   - **S.Haykin**- ***Neural Networks-A comprehensive foundation, 2003.***

CAPITOLUL 13.

PREZENTAREA SUCCINTĂ A UNOR TOOLBOX-URI MATLAB

MATLAB permite dezvoltarea unei familii de aplicaţii sub forma toolbox-urilor. Aceste toolbox-uri permit învăţarea şi aplicarea tehnologiilor specializate din diverse domenii. Sunt disponibile toolbox-uri pentru domenii cum ar fi: procesarea numerică a semnalelor, sisteme de conducere automată, reţele Neuronale, logică fuzzy, wavelet, simulare (SIMULINK), identificare, statistică, crearea de hărţi, procesarea imaginilor etc.
În continuare sunt prezentate pe scurt facilităţile oferite de câteva din aceste toolbox-uri.

13.1.Toolbox-ul Comunicaţii (Communications Toolbox)

Communications Toolbox este o colecţie de funcţii de calcul şi blocuri de simulare pentru cercetare, dezvoltare, proiectarea de sisteme şi simulare în domeniul comunicaţiilor. Toolboxul este proiectat atât pentru experţi cât şi pentru începători în domeniu, şi se bazează pe MATLAB şi Simulink.
Funcţii disponibile:
- Surse de date
- Surse codare/decodare
- Controlul erorilor de codificare
- Modulare/demodulare
- Filtre transmisie/recepţie
- Canale de transmisie
- Acces multiplu
- Sincronizare
- Utilitare

13.2.Toolbox-ul pentru Sisteme de Conducere Automată (Control System)

MATLAB-ul dispune de o colecţie bogată de funcţii utile atât inginerului automatist practician cât şi teoreticianului din domeniul teoriei sistemelor. Printre facilităţile oferite enumerăm: aritmetică complexă, valori proprii, găsirea rădăcinilor, inversări de matrici, Transformarea Fourier Rapidă etc.
Toolbox-ul Control System foloseşte structurile matriceale ale MATLAB şi reprezintă fundaţia MATLAB. Toolbox-ul este o colecţie de algoritmi (în principal sub forma fişierelor .m), care implementează proiectarea sistemelor de conducere, precum şi tehnici de analiză şi modelare.
Sistemele pot fi modelate în MATLAB ca funcţii de transfer, sub forma poli-zerouri sau prin reprezentarea de stare, ceea ce permite aplicarea tehnicilor clasice şi a celor moderne. Se poate lucra cu sisteme continue şi cu sisteme discrete şi pot fi efectuate conversii între diversele tipuri de modele. Toolbox-ul permite calculul şi trasarea răspunsurilor în domeniul timp şi domeniul frecvenţă, precum şi a locului rădăcinilor. De asemenea, se pot realiza alocări de poli, se pot implementa conducerea optimală şi estimatoare etc.

13.3.Toolbox-ul pentru Baze de Date (Database Toolbox)

Toolbox-ul Baze de Date permite importul şi exportul de date între MATLAB şi cele mai răspândite programe de baze de date. Cu acest toolbox se pot importa date din exterior, se utilizează capacităţile mari de calcul şi prelucrare analitică ale MATLAB, şi se exportă rezultatele înapoi în baza de date sau în altă bază de date.
Realizarea acestor operaţiuni se face astfel: toolbox-ul Database conectează MATLAB la o bază de date utilizând funcţiile MATLAB; datele sunt preluate de la baza de date ca şi caractere, sunt transformate în tipuri de date corespunzătoare şi sunt stocate în tablouri de tip celulă. În acest moment se pot folosi instrumentele MATLAB de lucru cu date. Se pot include funcţiile toolbox-ului în fişiere M-files. Pentru exportul datelor se utilizează în final funcţiile specializate MATLAB.
Toolbox-ul Database este furnizat împreună cu interfaţa grafică Visual Query Builder (VQB).

13.4. Toolbox-ul de Procesare a Semnalelor (Signal Processing Toolbox)

Toolbox-ul de Procesare a Semnalelor este o colecţie de instrumente construită în mediul de calcul numeric MATLAB. Toolbox-ul permite o mare varietate de operaţii de prelucrare a semnalelor, de la forme de undă la proiectarea filtrelor, modelare parametrică şi la analiza spectrală. Toolbox-ul furnizează două categorii de instrumente:

1. Funcţii de prelucrare a semnalelor de la linia de comandă
2. Intefeţe grafice utilizator pentru:
- Proiectarea interactivă a filtrelor
- Trasarea şi analiza semnalelor
- Analiză spectrală
- Aplicarea de filtre semnalelor
- Analiza filtrelor

13.5. Toolbox-ul DSP Blockset

Setul de blocuri de procesare numerică a semnalelor (DSP Blockset) este o colecţie de biblioteci de blocuri care se utilizează cu pachetul Simulink.
Bibliotecile DSP Blockset sunt proiectate special pentru prelucrarea numerică a semnalelor şi includ facilităţi cum ar fi filtrarea clasică, adaptivă, manipulări de matrici, algebră liniară, statistică, etc.
DSP Blockset extinde mediul Simulink prin furnizarea de componente şi algoritmi pentru sistemele DSP. Facilităţi:
• Operaţiuni bazate pe cadre
• Suport matriceal
• Filtrare adaptivă şi cu eşantionare multiplă
• Operaţiuni statistice
• Algebră liniară
• Estimarea parametrilor
• Facilităţi de timp real

13.6. Toolbox-ul Finanţe (Financial Toolbox)

MATLAB-ul împreună cu toolbox-ul de Finanţe furnizează un mediu de calcul integrat şi complet pentru analiză şi inginerie financiară. Toolbox-ul dispune de instrumente de analiză matematică şi statistică a datelor financiare şi instrumente de prezentare grafică a rezultatelor obţinute.

Facilităţi:

- Calcul şi analiză de preţ şi de producţie
- Realizează analize venituri, preţuri etc. compatibile SIA (Securities Industry Association)
- Analiza şi managementul portofoliilor
- Proiectarea şi evaluarea de strategii financiare
- Identificarea, măsurarea şi controlul riscului
- Analiza şi calculul fluxului de cash
- Analiza şi predicţia activităţii economice
- Crearea de instrumente financiare structurate
- Cercetare academică

13.7. Toolbox-ul de Procesare a Imaginilor (Image Processing Toolbox)

Toolbox-ul Image Processing este o colecţie de funcţii care extind posibilităţile MATLAB din domeniul prelucrării imaginilor. Toolbox-ul dispune de o mare varietate de operaţiuni de prelucrare a imaginilor, cum ar fi:

- Operaţii geometrice
- Operaţii de tip vecinătate
- Filtrare liniară şi proiectarea filtrelor
- Transformate
- Analiza şi îmbunătăţirea imaginilor
- Operaţii binare
- Operaţii asupra regiunii de interes

Multe dintre funcţiile toolbox-ului sunt fişiere M-files care constau în instrucţiuni MATLAB care implementează algoritmi specializaţi de prelucrare a imaginilor. Aceste instrucţiuni pot fi vizualizate cu comanda cunoscută:

```
type function_name
```

Posibilităţile toolbox-ului pot fi extinse prin crearea de fişiere proprii prin utilizarea funcţiilor disponibile în combinaţie cu alte toolbox-uri cum ar fi Signal Processing Toolbox şi Wavelet Toolbox.

13.8. Toolbox-ul Optimizare (Optimization Toolbox)

Acest toolbox este o colecţie de funcţii care includ rutine pentru o mare diversitate de optimizări:

- Minimizare neliniară fără constrângeri
- Minimizare neliniară cu constrângeri, inclusiv probleme de tip minimax şi probleme de minimizare semi-infinită
- Programare liniară şi pătratică
- Algoritmi neliniari de tipul celor mai mici pătrate
- Rezolvarea de sisteme de ecuaţii neliniare

Sunt disponibili şi algoritmi specializaţi pentru sisteme mari (large-scale problems).
Funcţiile toolbox-ului pot fi utilizate în combinaţie cu alte toolbox-uri sau cu Simulink.

13.9. Toolbox-ul pentru Sisteme de Putere (Power System Blockset)

Sistemele electrice de putere sunt combinaţii de circuite electrice şi de aparate electro-mecanice cum ar fi motoarele şi generatoarele. Inginerii care lucrează în acest domeniu trebuie să îmbunătăţească performanţele sistemelor de putere. Cerinţele de creştere drastică a eficienţei au determinat proiectanţii să folosească aparatură electronică şi sisteme sofisticate de conducere care necesită instrumente de analiză şi proiectare corespunzătoare, fiind absolut necesară înţelegerea fenomenelor (neliniare) prin simulare.
Power System Blockset a fost proiectat pentru furnizarea unor instrumente de proiectare care permit inginerilor să construiască rapid şi uşor modele care simulează sistemele de putere. Blocurile utilizează mediul Simulink şi permit construirea unui model cu proceduri simple de tip click and drag. Se poate trasa rapid topologia circuitelor electrice şi de asemenea se poate face o analiză a circuitelor care include interacţiunile cu sistemele mecanice, termice, de control.
Bibliotecile conţin modele ale aparaturii tipice pentru sistemele de putere, cum ar fi transformatoare, linii electrice, motoare, electronică de putere etc. Posibilităţile toolbox-ului pot fi testate prin rularea fişierelor demonstrative.

13.10. Toolbox-ul Stateflow (Diagrame de stare)

Stateflow este un produs multiplatformă, care poate rula pe sisteme Microsoft Windows XP, Windows 7 şi UNIX, şi care necesită MATLAB + Simulink.
Stateflow este un instrument grafic de proiectare şi dezvoltare pentru sisteme de conducere complexe şi pentru sisteme logice de supervizare. Cu ajutorul acestui toolbox se pot genera modele sub forma unor diagrame de stare dinamice ale unui sistem. Generarea de cod pentru elementele Simulink ale unui model Stateflow necesită pachetul suplimentar al Simulink-ului Real-Time Workshop.

13.11. Toolbox-ul de Statistică (Statistics Toolbox)

Toolbox-ul de Statistică dispune de instrumente care permit executarea de sarcini statistice uzuale, de la generarea de numere aleatoare până la proiectarea de experimente statistice şi controlul proceselor statistice (recomandăm studiu atent al capitolului 9 Elemente de probabilităţi şi Statistică Matematică).
Există două categorii de instrumente:
• Construcţia de funcţii statistice şi probabilistice
• Instrumente grafice interactive
Prima categorie constă în funcţii care pot fi apelate din linia de comandă sau din aplicaţii proprii.
A doua categorie constă în instrumente interactive care permit accesul la funcţii prin intermediul unei interfeţe grafice utilizator (GUI), care furnizează un mediu adecvat pentru funcţii de predicţie, interpolare, probabilistice etc.

13.12. Toolbox-ul pentru Calcul Simbolic (Symbolic Math Toolbox)

Acest toolbox completează facilităţile grafice şi numerice ale MATLAB-ului cu diverse alte tipuri de facilităţi matematice, care permit efectuarea de calcule simbolice (17) .
Maşina de calcul utilizată este nucleul de la Maple. Există de fapt două toolbox-uri:

- unul de bază, care este o colecţie de peste o sută de funcţii MATLAB care permit accesul la Maple,
- un toolbox extins care permite accesul extins la multiple pachete Maple (non-grafice, facilităţi de programare, proceduri utilizator etc.).

Facilităţi:

Calcul matematic:	Diferenţieri, integrări, limite, sume, serii Taylor
Algebră liniară:	Inverse, determinanţi, valori proprii, forme canonice ale matricelor.
Simplificări:	Metode de simplificare a expresiilor algebrice
Rezolvarea ecuaţiilor:	Soluţii simbolice şi numerice ale ecuaţiilor algebrice şi diferenţiale
Aritmetică cu precizie variabilă: specificată	Evaluarea numerică a expresiilor matematice cu precizie
Transformări:	Fourier, Laplace, Transformarea z şi inversele lor
Funcţii matematice speciale:	Funcţii speciale ale matematicilor aplicate clasice

Toolbox-ul pentru Calcul Simbolic defineşte un nou tip de date : obiecte simbolice care se pot crea cu ajutorul funcţiei `sym`. Aceste obiecte simbolice sunt utilizate pentru a reprezenta variabile simbolice, expresii, şi matrici. Spre exemplu pentru a calcula derivatele unei funcţii se poate utiliza funcţia `diff`.

Exemplu

```
>> x=sym('x');
>> f=exp(cos(x))*sin(x);
>> diff_f=diff(f,x)% calculeaza derivata de ordin I
diff_f =
exp(cos(x))*cos(x) - exp(cos(x))*sin(x)^2
>> diff(f,x,2)% calculeaza derivata de ordin II
ans =
exp(cos(x))*sin(x)^3 - exp(cos(x))*sin(x) -
3*exp(cos(x))*cos(x)*sin(x)
>> diff(f,x,5)%calculeaza derivata de ordin V
ans =
exp(cos(x))*cos(x) + 15*exp(cos(x))*cos(x)^2 +
15*exp(cos(x))*cos(x)^3 - 16*exp(cos(x))*sin(x)^2 +
20*exp(cos(x))*sin(x)^4 - exp(cos(x))*sin(x)^6 -
45*exp(cos(x))*cos(x)^2*sin(x)^2 - 75*exp(cos(x))*cos(x)*sin(x)^2 +
15*exp(cos(x))*cos(x)*sin(x)^4
```

Pentru mai multe detalii se poate folosii comanda:

```
>> help sym/diff
 diff   Differentiate.
    diff(S) differentiates a symbolic expression S with respect to
its
    free variable as determined by SYMVAR.
    diff(S,'v') or diff(S,sym('v')) differentiates S with respect to
v.
    diff(S,n), for a positive integer n, differentiates S n times.
    diff(S,'v',n) and diff(S,n,'v') are also acceptable.
    Examples;
       x = sym('x');
       t = sym('t');
       diff(sin(x^2)) is 2*x*cos(x^2)
       diff(t^6,6) is 720.

    See also sym/int, sym/jacobian, sym/symvar.
```

Calculul integralelor definite.

Exemplu:

```
>>  g=exp(-2*x)*sin(3*x);
>> int_g=int(g,x)
int_g =
-(3*cos(3*x) + 2*sin(3*x))/(13*exp(2*x))
```

Dezvoltări în serie Taylor:

Exemplu :

```
>> clear,syms x, Tay_expx=taylor(exp(x),5,x,0)
Tay_expx =
x^4/24 + x^3/6 + x^2/2 + x + 1
>> pretty(Tay_expx)
   4    3    2
   x    x    x
  -- + -- + -- + x + 1
```

Limite

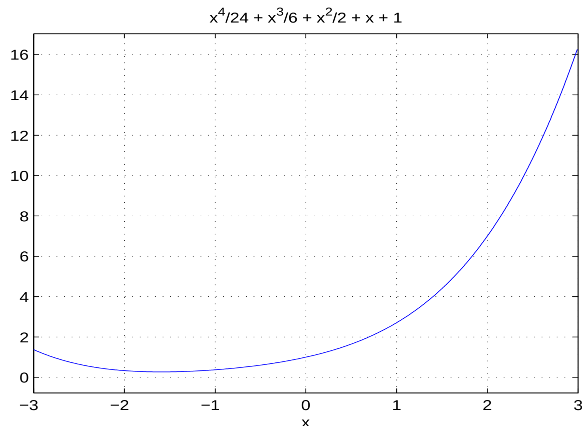
```
 >>L=limit(asin(x)/x,x,0)
L =
1
```

Pentru a reprezenta expresii simbolice se poate utiliza funcţia `ezplot` .

Exemplu

```
>>  ezplot(Tay_expx,[-3,3]);grid on
```

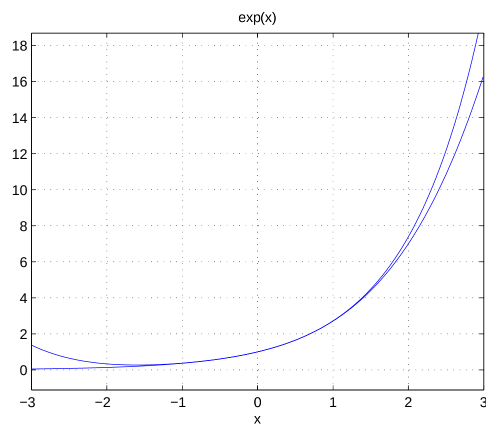
Se obţine graficul:



Dacă dorim a reprezenta două grafice în aceaşi figură:

```
>>ezplot(Tay_expx,[-3,3]);grid on;hold on
>>ezplot(exp(x),[-3,3]);grid on
```

Se obţine graficul:



Rezolvarea sistemelor liniare şi neliniare.

Exemplu

```
>> clear,syms x y
>> eq1='y=2*exp(x)';eq2='3-x^2';
>>  [x,y]=solve(eq1,eq2,x,y)
x =
   3^(1/2)
  -3^(1/2)
y =

  2*exp(3^(1/2))
  2/exp(3^(1/2))
```

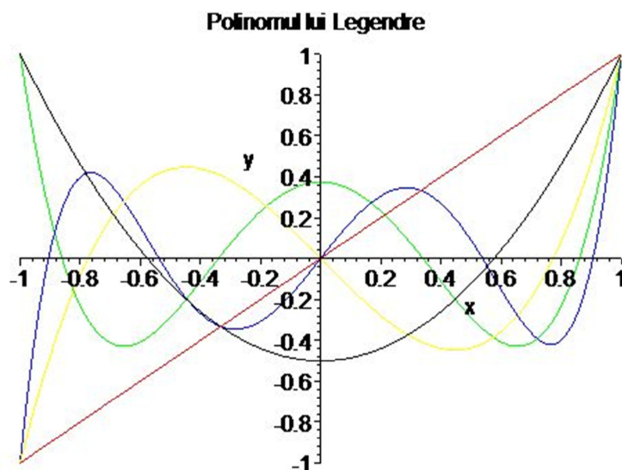
O altă modalitate de lucru este lansarea în execuţie a notebook-ului `Mupad` astfel :

```
>>mupad
```

Apare o fereastra delimitata în zona de lucru de simbolul [unde se introduce codul Maple:

```
> restart;
with(orthopoly):with(CodeTools):with(plots):
> S:=(x,k,a,b)->P(k,((b-a)*x+a+b)/2):
> a:=-1:b:=1:k:=5:
> p5:=plot(S(x,k,a,b),x=-1..1,y=-
1..1,color=[BLUE],title="Polinomul lui Legendre"):
> k:=4:
> p4:=plot(S(x,k,a,b),x=-1..1,y=-
..1,color=[GREEN],title="Polinomul lui Legendre"):
> k:=3:
> p3:=plot(S(x,k,a,b),
x=-1..1,y=-1..1,color=[YELLOW],title="Polinomul lui
Legendre"):
> k:=2:
> p2:=plot(S(x,k,a,b),x=-1..1,y=-
1..1,color=[BLACK],title="Polinomul lui Legendre"):
> k:=1:
> p1:=plot(S(x,k,a,b),x=-1..1,y=-
1..1,title="Polinomul lui Legendre "):
> plots[display]({p1,p2,p3,p4,p5});
```

În urma execuției programului va rezulta reprezentarea grafică a **polinoamelor lui Legendre** de grade 1,2,3,4,5:



Comanda:

```
>> mupadwelcome
```

Are ca effect deschiderea unei ferestre de lucru interactive.

Comanda:
>> `mupaddemo`
Are ca effect dechiderea unei ferestre de tip demo .

13.13. Toolbox–ul de ecuaţii cu derivate parţiale (PDE toolobox)

Modul de utilizare este următorul (32):

1. cu comanda `>> pdetool` se deschide foaia de lucru `PDE Toolbox`.
2. din meniul `options` se activează `grid` şi apoi `în submeniul application` se selectează generic `scalar` ca tip de ecuaţie.
3. Din meniul `Draw` se activează `draw mode` şi apoi `polygon`.
4. Pe foaia de lucru `PDE toolbox` se trasează cu mouse-ul frontiera domeniului de calcul .
5. Din domeniul `Boundary` se activează `boundary mode` şi apoi `specify boundary conditions;` în fereastra de dialog care apare se selectează tipul de condiţie la limită *Dirichlet sau Neuman* precum şi parametrii respectivi. Condiţiile la limită se pot da şi individual pe fiecare segment de frontieră prin click cu mouse-ul pe acel segment.
6. Din meniul `PDE` se activează `PDE mode` şi apoi `PDE` specification`;` în fereastra de dialog care apare se alege de exemplu tipul de ecuaţie `eliptic` ceeace corespunde la `–div(c*grad(u)+a*u=f` şi coeficienţii: `c=1, a=0, f=sin (x).*sin(y).`
7. Din meniul `Mesh` se activează `mesh mode` şi apoi `initialize mesh` ceea ce generează o triangulaţie de pornire care apare pe desen.
8. Din meniul `Solve` se selectează `Parameters` şi apoi în fereastra de dialog care apare se activează `Adaptive mode.` Această opţiune permite rafinarea succesivă a triangulaţiei în funcţie de calităţile soluţiei aproximate. Tot din meniul `Solve,` prin activarea lui `Solve PDE` se trece la rezolvarea numerică a problemei introduse cu metoda elementului finit, prin rafinări succesive ale tringulaţiei iniţiale până la îndeplinirea unui criteriu de stop.
9. Din meniul `Plot` se activează `Parameters,` iar apoi în fereastra de dialog care apare se selectează `color şi countour` care determină modul de reprezentare grafică a soluţiei.

Există numeroase alte opţiuni care sunt bine prezentate în paginile de `Help` ale produsului şi în exemplele demonstrative.

CAPITOLUL 14.

PACHETUL DE MODELARE ŞI SIMULARE SIMULINK®

SIMULINK® [19] este un pachet software pentru modelarea, simularea şi analiza sistemelor dinamice. Pot fi modelate sisteme liniare şi neliniare, continue, discrete, hibride, cu mai multe perioade de eşantionare

SIMULINK® furnizează o interfaţă grafică utilizator (GUI) [14] pentru crearea modelelor sub forma unor diagrame construite din blocuri, pe baza unor tehnici de tip click-and-drag realizate cu mouse-ul. Astfel, trasarea diagramelor este simplă şi intuitivă, aproape la fel de simplă ca trasarea acestor diagrame direct pe hârtie.

În plus, se evită formularea matematică laborioasă (sistemele dinamice sunt de regulă descrise de ecuaţii diferenţiale sau cu diferenţe).

SIMULINK® dispune de o bibliotecă vastă de surse, receptoare, componente liniare şi neliniare, conectori etc. pe baza cărora se pot trasa diagrame şi construi blocuri proprii.

Modelele realizate în SIMULINK® sunt ierarhice. Se poate vizualiza modelul de nivel înalt, iar la efectuarea unui dublu click pe blocul respectiv se coboară nivel după nivel astfel încât se pot observa toate detaliile de construcţie şi de organizare ale modelului.

După crearea unui model se pot realiza simulări apelând la diverse metode de integrare din meniurile SIMULINK® şi/sau utilizând comenzi MATLAB®. Prin utilizarea unor blocuri de tip osciloscop sau diverse dispozitive de afişare se pot observa rezultatele chiar în timpul simulării. De asemenea se pot schimba valorile unor parametri şi se poate observa imediat efectul acestor modificări. Rezultatele obţinute se pot transporta în workspace-ul MATLAB® pentru prelucrări şi vizualizări ulterioare.

14.1. Rularea unui model SIMULINK® demonstrativ

14.1.1. Rularea modelului

Pentru a analiza modul de lucru cu SIMULINK® se poate apela la rularea unor programe (modele) demonstrative.

Unul din programele demo este modelul termodinamic al unei case.

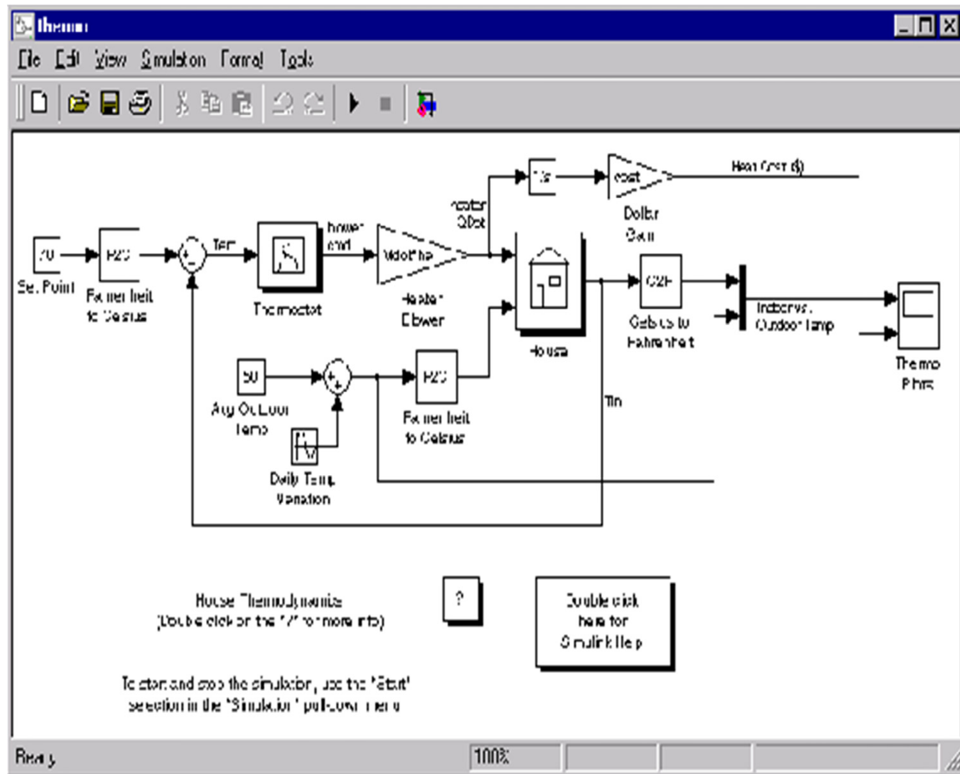
Exemplul este luat de pe site-ul: http://www.mathworks.com

MATLAB SIMULINK ® - Simulation and Model Based Design

Pentru rularea programului, trebuie parcurşi următorii paşi:

Pas 1. Se startează MATLAB®.

Pas 2. Se rulează demonstraţia tastând thermo în fereastra de comandă MATLAB® sau se tastează comanda demo şi se alege programul demonstrativ din meniul care apare. Aceste comenzi startează SIMULINK® şi creează o fereastră model care conţine modelul respectiv.



La deschiderea modelului (extensiile fişierelor SIMULINK sunt .mdl) SIMULINK-ul deschide un bloc de tip osciloscop cu două ecrane (temperatură interioară/exterioară - Indoor vs. Outdoor Temp. şi costul încălzirii - Heat Cost ($)).

3. Pentru startarea simulării se activează meniul Simulation şi se alege comanda Start command (sau se activează direct butonul Start din bara de instrumente). O dată cu startarea simulării sunt plotate evoluţiile temperaturii interioare şi exterioare, ca şi costul cumulat al încălzirii.

4. Pentru oprirea simulării se alege comanda Stop din meniul Simulation (sau butonul Pause din bara de instrumente).

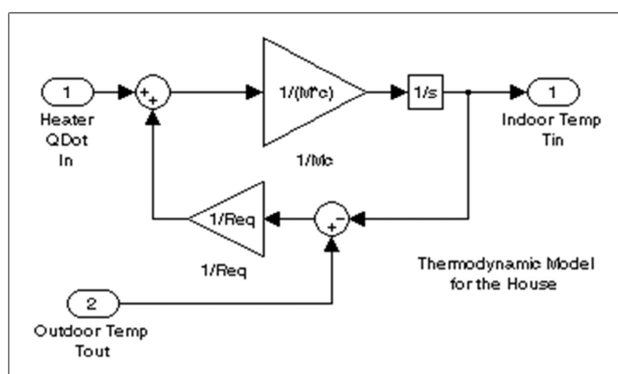
5. Atunci când se doreşte terminarea rulării programului se închide modelul alegând Close din meniul File.

14.1.2. Descrierea modelului

Programul modelează sistemul termodinamic al unei case folosind o reprezentare simplă. Temperatura de referinţă este setată la 70 grade Fahrenheit (aprox. 21 grade Celsius). Temperatura din casă este influenţată de temperatura exterioară, care poate fi variată sub formă sinusoidală (amplitudine 15 grade F, temperatura de bază 50 grade F), variaţie care simulează fluctuaţiile temperaturii din exterior din timpul zilei.

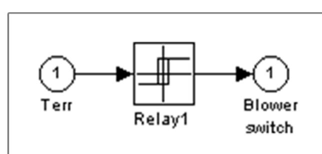
Sunt utilizate subsisteme care fac modelul simplu şi configurabil (un subsistem este un bloc alcătuit dintr-un grup de blocuri conectate).

Modelul conţine 5 subsisteme: Thermostat, House şi trei Convertoare de temperatură (Temp Convert), din care 2 convertesc Fahrenheit în Celsius şi unul Celsius în Fahrenheit.
Efectuarea unui dublu click pe blocul House permite vizualizarea blocurilor componente ale subsistemului.



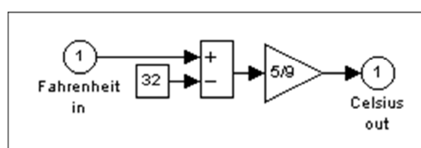
House subsystem

Subsistemul Thermostat este de tip releu şi determină pornirea sau oprirea sistemului de încălzire. Se pot vedea blocurile componente la efectuarea unui dublu click pe subsistem.



Thermostat subsystem

Subsistemele de conversie a temperaturii au o structură asemănătoare:



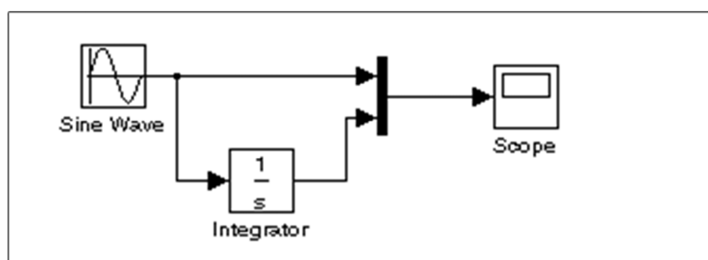
Fahrenheit to Celsius conversion (F2C)

Alte demonstraţii
Din fereastra bibliotecilor SIMULINK® pot fi rulate şi alte demonstraţii care pun în evidenţă diverse concepte şi tehnici de modelare din diverse domenii. Pentru rularea din fereastra bibliotecilor SIMULINK® se procedează astfel:

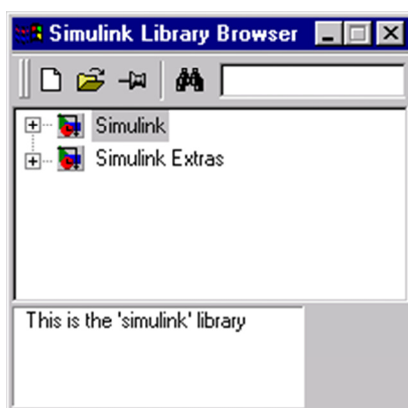
1. Se tastează **simulink3** în fereastra de comandă MATLAB; va apare fereastra bibliotecilor SIMULINK®
2. Se execută dublu click pe icon-ul ***Demos***. Va apare fereastra demo a MATLAB® -ului, care conţine câteva modele SIMULINK® interesante.

14.2. Crearea modelelor SIMULINK®

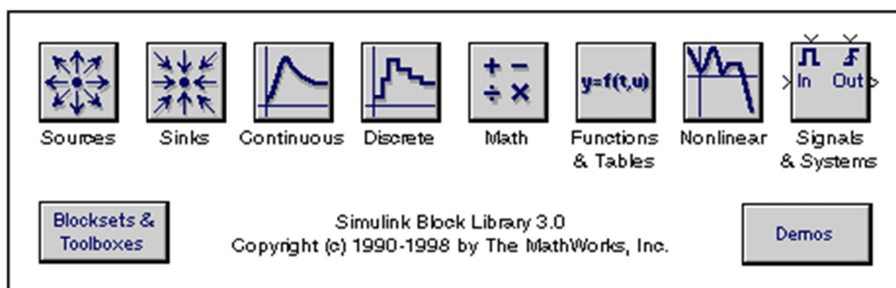
Tehnica de creare a unor modele SIMULINK® poate fi ilustrată cel mai bine prin exemple.
Modelul prezentat în continuare integrează un sinus şi afişează atât rezultatul cât şi unda sinusoidală de la intrare. Schema bloc a modelului este următoarea:



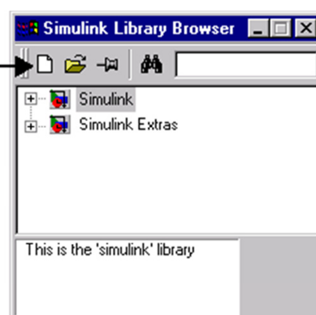
Pentru a genera modelul se tastează mai întâi simulink în fereastra de comandă MATLAB® Pe sistemele de operare de tip Windows va apare Browser-ul bibliotecilor SIMULINK®



Pe sistemele UNIX, va apare fereastra bibliotecilor SIMULINK.



Pentru a genera un model nou pe sisteme UNIX se selectează Model din submeniul New al meniului File. Pe sisteme Windows se selectează butonul ***New Model*** din bara de instrumente a Browser-ului de Biblioteci.

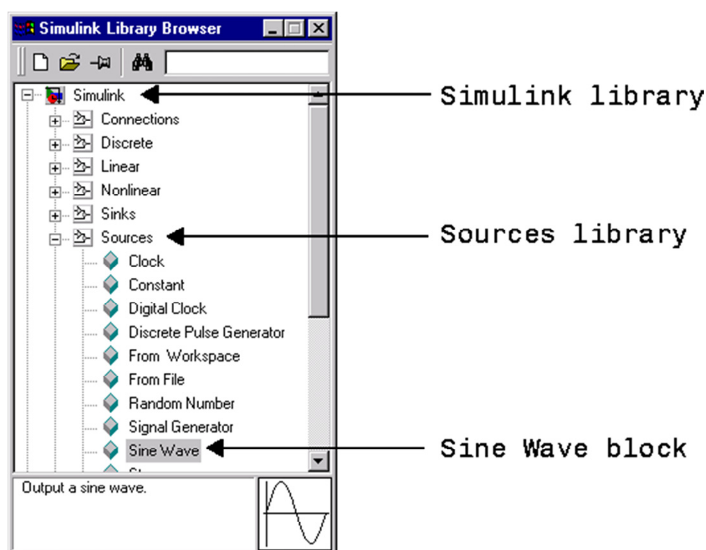


SIMULINK® va deschide o fereastră pentru un model nou.
Pentru construcţia modelului vor fi necesare blocuri din următoarele biblioteci Simulink:

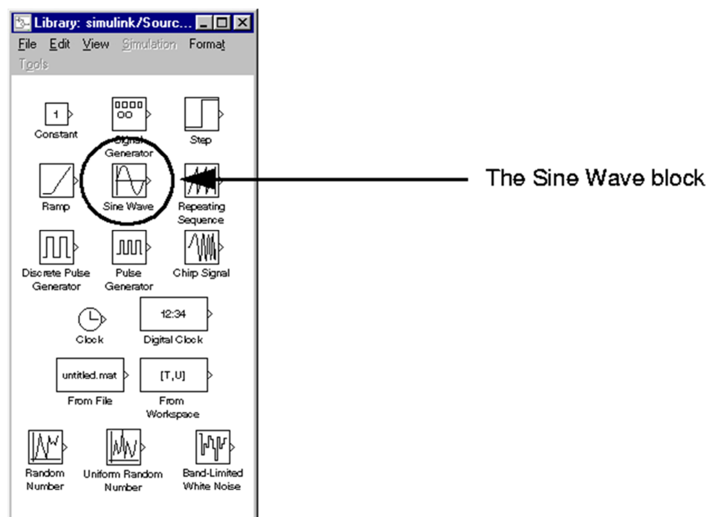
- Biblioteca de surse (blocul Sine Wave)
- Biblioteca de receptoare (blocul Scope)
- Biblioteca de sisteme continue (blocul Integrator)
- Biblioteca Signals & Systems (blocul Mux)

Pentru copierea blocului Sine Wave se utilizează Browser-ul de bibiloteci: întâi se expandează arborele de biblioteci (prin click pe nodul Simulink şi apoi click pe nodul surse) astfel încât să fie afişate blocurile din biblioteca de surse. Apoi se selectează blocul Sine Wave (click).

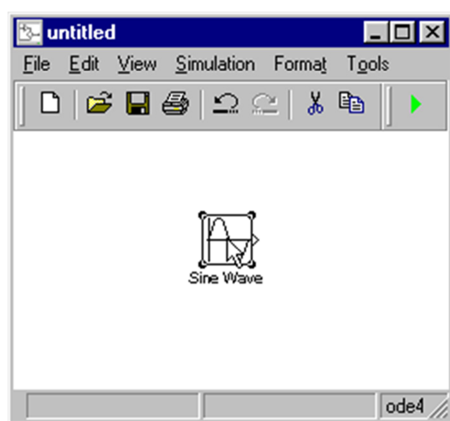
Fereastra Browser-ului de biblioteci va arăta astfel:



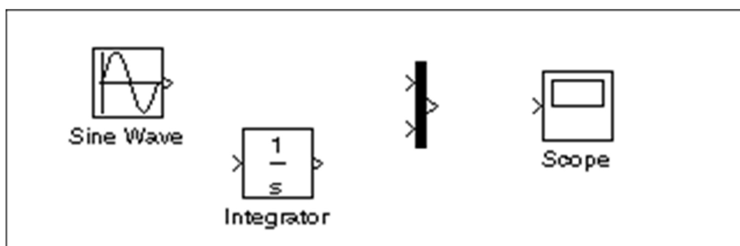
La pasul următor se trage (ţinând apăsat butonul din stânga al mouse-ului) blocul Sine Wave din browser şi i se dă drumul în fereastra modelului. Simulink va face o copie a blocului Sine Wave în punctul indicat.
Se poate proceda asemănător pentru copierea blocului Sine Wave din biblioteca de surse deschisă din fereastra de biblioteci Simulink (pe sisteme Windows se poate deschide fereastra de biblioteci din Browser prin click din butonul drept al mouse-ului şi apoi click pe Open Library).



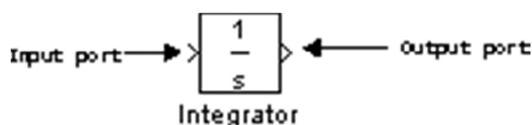
Ca şi în cazul utilizării browser-ului se trage blocul Sine Wave din biblioteca de surse în fereastra modelului (drag and drop):



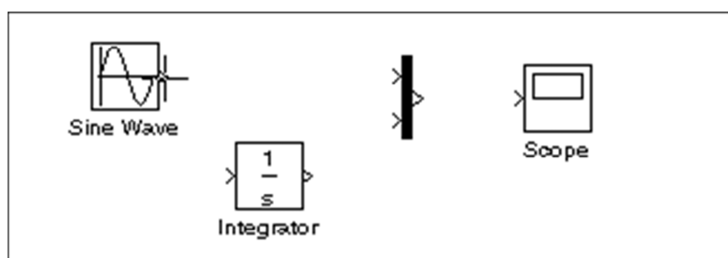
Se procedează în mod asemănător şi cu copierea celorlalte blocuri din bibliotecile corespunzătoare în fereastra modelului. Se poate deplasa cu uşurinţă orice bloc prin tragerea cu mouse-ul sau prin selectare şi deplasare cu tastele săgeţi. După copierea tuturor blocurilor necesare în fereastra de lucru, modelul trebuie să arate ca în figura următoare:



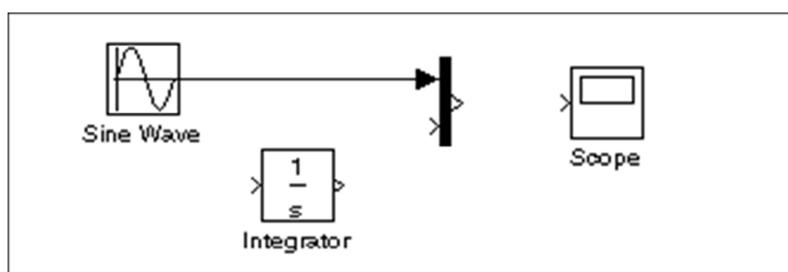
La o examinare atentă a simbolurilor de reprezentare a blocurilor se vor observa săgeţi care indică intrările sau ieşirile din blocuri: dacă simbolul > este orientat spre ieşirea blocului atunci este un port de ieşire (output port) iar dacă simbolul este îndreptat spre bloc este un port de intrare (input port). Un semnal circulă de la un port de ieşire al unui bloc spre un port de intrare al altui bloc printr-o linie de conectare, atunci când blocurile sunt conectate, simbolurile porturilor dispar.



Pentru conectarea blocurilor se trece întâi la conectarea blocului Sine Wave la prima intrare (de sus) a blocului Mux. Pentru aceasta se poziţionează pointerul mouse-ului deasupra portului de ieşire al blocului Sine Wave. În acest moment forma pointerului se schimbă şi devine de tip cruce (cursor).



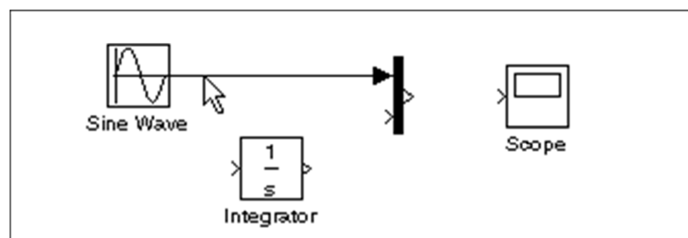
Se ţine apăsat butonul stânga al mouse-ului şi se deplasează cursorul până la intrarea de sus a blocului Mux.
Urmează eliberarea butonului mouse-ului şi se observă cum blocurile au fost conectate.



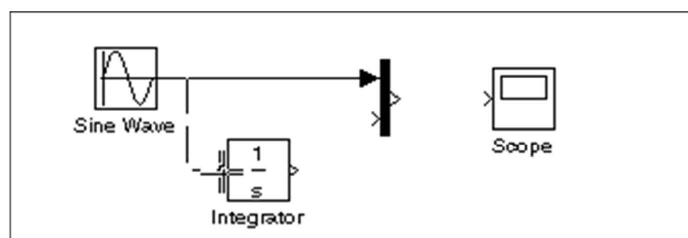
În afara liniilor care conectează ieşirile unor blocuri la intrările altora există şi linii de branşare a unor linii la intrările unor blocuri (se poate observa în modelul prezentat iniţial). O astfel de linie este utilizată pentru conectarea ieşirii din blocul Sine Wave şi la blocul Integrator (există deja conexiunea la blocul Mux).

Pentru a efectua această conexiune se procedează astfel:

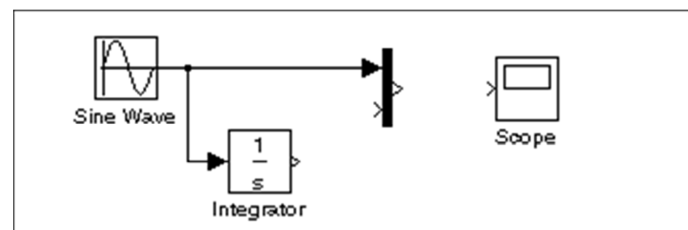
1. Se poziţionează pointerul pe linia dintre blocurile Sine Wave şi Mux.



2. Se apasă şi se ţine apăsată tasta Ctrl. Se apasă butonul mouse-ului şi apoi se trage până la intrarea în blocul Integrator sau până deasupra acestui bloc.

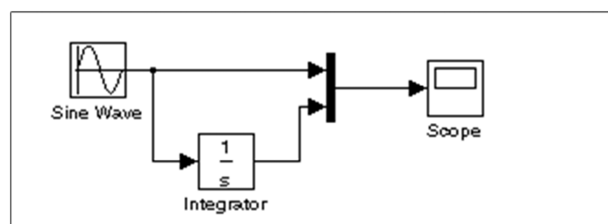


3. Se eliberează butonul mouse-ului şi se observă cum apare o linie de branşare până la portul de intrare în blocul Integrator.



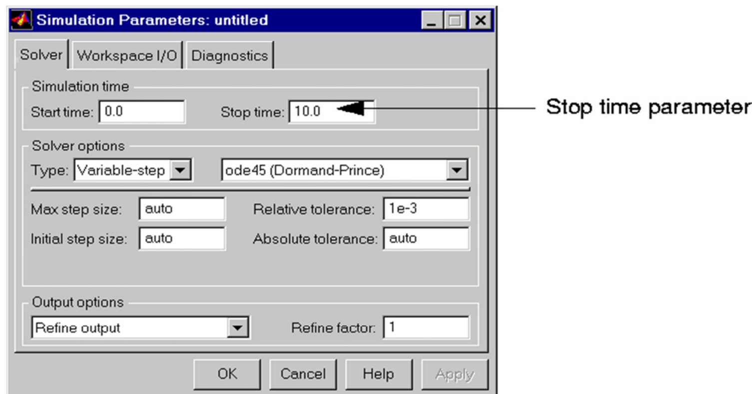
Se procedează conform indicaţiilor şi se efectuează toate conectările necesare.

Modelul va trebui să arate în final astfel:

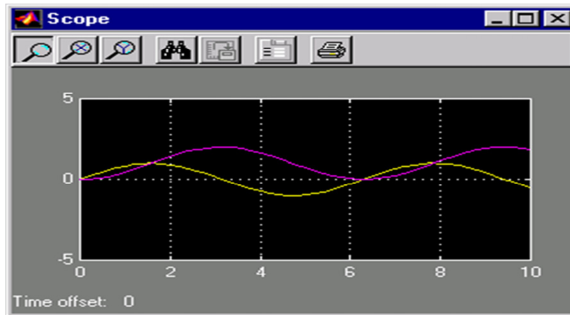


14.3. Rularea simulărilor în SIMULINK®

După încheierea procedurii de construcţie a modelului, trebuie rulată o simulare pentru aprecierea corectitudinii modelului şi pentru obţinerea rezultatelor cerute.
Pentru aceasta se deschide mai întâi blocul osciloscopului (Scope), pentru a vizualiza evoluţia mărimilor modelului. Păstrând fereastra osciloscopului deschisă se va seta SIMULINK® pentru rularea unei simulări timp de 10 secunde. Pentru aceasta, parcurgem următorii paşi:
1. Setăm parametrii simulării prin alegerea submeniului Parameters din meniul Simulation. În fereastra de dialog care apare vom seta parametrul Stop time la 10.0 (valoare implicită).
2. Închidem fereastra de dialog Simulation Parameters prin click pe butonul Ok. Simulink va aplica parametrii şi va închide fereastra de dialog.



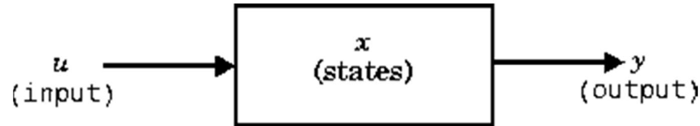
3. Se selectează Start din meniul Simulation şi se observă curbele afişate în fereastra osciloscopului.



4. Simularea se va opri atunci când se ajunge la finalul timpului de rulare indicat în fereastra de dialog Simulation Parameters sau la selectarea opţiunii Stop din meniul Simulation (sau din bara de instrumente).
Pentru salvarea modelului se alege Save din meniul File şi se introduce un nume de fişier şi se alege directorul unde dorim să salvăm modelul (fişierul va avea automat extensia .mdl).
Pentru încheierea lucrului se selectează Exit MATLAB sau se tastează quit în fereastra de comandă a MATLAB-ului.

14.4. Modul de lucru al unui program SIMULINK®

Fiecare bloc dintr-un model Simulink are următoarele caracteristici generale: un vector de intrare, u, un vector de ieşire, y, şi un vector de stare, x:



Vectorul de stare poate consta din stări continue, stări discrete sau combinaţii ale acestora. Relaţiile matematice dintre aceste mărimi (intrări, ieşiri, stări) sunt exprimate prin ecuaţii de tipul:

$$y = f_0(t,x,u) \qquad iesirea$$

$$x_{d_{k+1}} = f_u(t,x,u) \qquad actualizare$$

$$x_c' = f_d(t,x,u) \qquad derivata$$

$$unde \quad x = \begin{bmatrix} x_c \\ x_{d_k} \end{bmatrix}$$

Simularea constă în două faze: ***iniţializare*** şi **simulare propriu- zisă.**

Faza de iniţializare înseamnă parcurgerea următoarelor etape:

1. Blocul parametrilor este trecut în MATLAB pentru evaluare. Valorile numerice rezultate sunt folosite ca parametri actuali (curenţi).

2. Este parcursă ierarhia modelului. Fiecare subsistem care nu este un subsistem executat condiţionat este înlocuit prin blocurile componente.

3. Blocurile sunt sortate în ordinea în care este necesară actualizarea lor. Algoritmul de sortare întocmeşte o listă astfel încât orice bloc nu este actualizat până când blocurile care furnizează intrările acestuia nu sunt actualizate. În timpul derulării acestei etape sunt detectate buclele algebrice.

4. Conexiunile dintre blocuri sunt verificate pentru asigurarea compatibilităţii ieşire-intrare.

Urmează faza de simulare propriu-zisă. Modelul este simulat prin integrare numerică. Calculul derivatelor se face în doi paşi. Prima dată ieşirea fiecărui bloc este calculată în ordinea determinată de algoritmul de sortare. La al doilea pas, pentru fiecare bloc se calculează derivatele în funcţie de timp, intrări şi stări. Vectorul derivatelor rezultat este returnat algoritmului de rezolvare de tip ODE, care îl utilizează pentru calculul noului vector de stare pentru momentul de timp următor. O dată ce noul vector de stare este calculat, blocurile sunt actualizate.

Faza de simulare propriu zisă constă în lansarea simulink-ului astfel construit.

Bibliografie Extinsă

1. Blaga P. - *Statistică prin Matlab,* Editura Presa Universitară Clujeană, Cluj-Napoca, 2002.

2. Burden L., Faires J.D.-*Numerical Analysis,* - PWS Kent, Boston, 1986.

3. Chow C.Y-*A Introduction to Computational Fluid Mechanics,*- John Wiley and Sons,1979.

4. Coman G.-*Numerical Analysis,* - Libris, Cluj-Napoca, 1985.

5. Dormand J.-*Numerical Methods for Differential Equations, A Computational Approach,*- CRC Press, Boca Raton New-York, 1996.

6. Guckenheimer J., Holmes P. – *Nonlinear Oscillations, Dynamical Systems and Bifurcations of Vector Fields,* – 3rd ed., Springer-Verlag, New York,1990.

7. Ghinea M., Fireţeanu V. - *Matlab, Calcul numeric, Grafică, Aplicaţii,* Editura Teora, 2006.

8. Gladwell I., Shampine L.F., Thomson S. - *Solving ODEs with MATLAB,* Cambridge University Press, 2003.

9. Gheorghiu C.I. – *Metode numerice pentru sisteme dinamice* – Casa Cărţii de Ştiinţă Cluj-Napoca, 2006.

10. Kohr M. - *Special Chapter of Mechanics*- Cluj University Press, 2005.

11. Kuzneţov Y.A. - *Elements of Applied Bifurcation Theory,*- Springer Verlag 2004.

12. Li F., Kuang Y. , Li B.- *Analysis of IVGT glucose-insulin interaction models with time delay ,*- Discret and Continuous Dynamical System Series B , Vol I, Nr.1, pag: 103-124, 2001.

13. Leonard N.E., Levine W.S – *Using MATLAB to analyze and design Control Systems*, Addison-Wesley Publ., SUA, 1995.

14. Marchand P. – *Graphics and GUIs with MATLAB,* CRC Press, SUA, 2003.

15. The Mathworks – *MATLAB 7. Getting Started Guide, 2008*- Minor revision for MATLAB 7.7 (Release 2008b).

16. The Mathworks, Inc. – *MATLAB 7.*- Programming Fundamentals, 2008., Revised for Version 7.7 (Release 2008b).

17. The Mathworks, Inc. – *MATLAB 7.* - Symbolic Math Toolbox 5, 2008., Revised for Version 7.7 (Release 2008b).

18. Micula G., Micula S.. – *Handbook of Splines* – Kluwer Academic Publishers, Netherland, 1999.

19. Nakamura S . - *Numerical Computing and Graphic Vizualization in MATLAB,* - Prentice Hall, Englewood Clifs, NJ, 1996.

20. Opriş D. Mircea G. – *The numerical Integrators of Dynamical Systems*- Seminar of Differential Dynamical Systems , Universitatea de Vest Timişoara, 1998.

21. Petrila T, Trif D - *Metode numerice şi computaţionale în dinamica fluidelor*- Editura Digital Data Cluj, 2002.

22. Pop N. D - *Aspects concerning some numerical methods for solving boundary value problems,* Cluj University Press, 2011.

23. Postelnicu T., Tăutu P. - *Metode matematice în medicină şi biologie,* Editura tehnică, Bucureşti, 1971.

24. Quarteroni A., Sacco R., Saleri F. - *Numerical Mathematics,* - Springer, New York, Berlin , Heidelberg, 2000.

25. Shampine L.F., Gladwell I, Thomson S. – *Solving ODEs with Matlab, Instructor's Manual* Cambridge University Press, 2002.

26. Stuart A.M, Humphries, A.R. – *Dynamical Systems and Numerical Algebra* – Cambridge University Press, 1996.

27. Syebehely, V . - *Theory of Orbits*- Academic Press, 1967.

28. Tassi, Ph.-*Metodes statistiques*- Second edition, Economica, Paris, 1989.

29. Trîmbiţaş R.T- *Numerical Analysis in Matlab*, Cluj University Press, 2009.

30. Trif, D., - *Metode Numerice pentru ecuaţii diferenţiale şi Sisteme dinamice,* Transilvania Press, Cluj 1997.

31. Verhulst F. - *Nonlinear Differential Equattion*- Springer Verlag, 1985.

32. Udrişte C.- *Geometric Dynamics,*- Kluwer Academic Publishers, 2000.

33. White R.E. - *Computational Mathematics, Models, Methods, and Analysis with Matlab and MPI,* Champan &Hall/CRC, Boca Raton, London, New York, Washington, D.C., 2004.

34. Wiggins S.- *Introduction to Applied Nonlinear Dynamical Systems and Chaos,* Springer-Verlag, 1990.

35. Wilson H.B.,Turcotte L.H., Halpern D.-*Advanced Mathematics and Mechanics Application Using Matlab,*-third ed.Champan &Hall/CRC, Boca Raton, London, New York, Washington, D.C., 2003.

36. Wilkinson J.H- *The Algebraic Eingenvalue Problem, -* Claredon Press, Oxford, 1965.

37. Xu R., - *Global analysis in a pure-delayed type Lotka-Volterrra system with one predator and twopreys.*-Contemporary differential equations and applications, 81-90, Nova Sci. Publ., Huappauge, NY, 2004

38. Zhang G., Cheng S.S, - *Positive periodic solutions of coupled delay differential systems,*Taiwannese Journal of Mathematics, 8 (2004), no 4., pp 639-652.

Adrese de web utile:

http://www.ma.ma.ac.uk/MCCM/MCCM.html

http://www.netlib.org/templates

http://www.3d-meier.de/tut3/Seite0.html

http://mathworks.com/moler

http://www-users.cs.umn.edu/~saad/books.html